TURING

猎豹移动CEO **傅盛**
360手机助手高级总监 **陶伟华**
联合推荐

[美] Chris Vander Mey ○著
刘亦舟○译

谷歌和亚马逊如何做产品

SHIPPING GREATNESS

Google Maps产品经理与您分享打造极致产品的经验

PRACTICAL LESSONS ON BUILDING AND LAUNCHING OUTSTANDING SOFTWARE LEARNED ON THE JOB AT GOOGLE AND AMAZON

O'REILLY®
人民邮电出版社
POSTS & TELECOM PRESS

数字版权声明

TURING

[美] Chris Vander Mey◎著
刘亦舟◎译

谷歌和亚马逊如何做产品

SHIPPING GREATNESS

Google Maps 产品经理与您分享打造极致产品的经验

人民邮电出版社
POSTS & TELECOM PRESS

图书在版编目 (CIP) 数据

谷歌和亚马逊如何做产品 / (美) 梅 (Mey,C.V.) 著 ; 刘亦舟译 . -- 北京 : 人民邮电出版社 , 2014.6

书名原文：Shipping greatness:practical lessons on building and launching outstanding software, learned on the job at Google and Amazon

ISBN 978-7-115-34931-6

Ⅰ. ①谷… Ⅱ. ①梅… ②刘… Ⅲ. ①网络企业－产品管理－经验－美国②电子商务－商业企业管理－产品管理－经验－中国 Ⅳ. ① F279.712.444 ② F724.6

中国版本图书馆 CIP 数据核字（2014）第 043562 号

内 容 提 要

本书是谷歌和亚马逊一线产品经理的经验总结，分为两大部分，共 13 章，系统阐述世界一流公司如何开发极致产品，让读者信心十足地迎接产品、方案、项目和工程管理的挑战。

本书适合 IT 行业产品经理以及所有团队主管和对管理感兴趣的人士。

♦ 著　　　　[美] Chris Vander Mey
译　　　　刘亦舟
责任编辑　刘美英
执行编辑　李岩俨
责任印制　焦志炜

♦ 人民邮电出版社出版发行　　北京市丰台区成寿寺路 11 号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京　　　　　　　　印刷

♦ 开本：720 × 960　1/16
印张：13
字数：224 千字　　　　2014 年 6 月第 1 版
印数：1–4 000 册　　　2014 年 6 月北京第 1 次印刷

著作权合同登记号　图字：01-2012-9298 号

定价：49.00 元

读者服务热线：(010)51095186 转 600　印装质量热线：(010)81055316

反盗版热线：(010)81055315

广告经营许可证：京崇工商广字第 0021 号

O’Reilly Media, Inc. 介绍

O’Reilly Media 通过图书、杂志、在线服务、调查研究和会议等方式传播创新知识。自 1978 年开始，O’Reilly 一直都是前沿发展的见证者和推动者。超级极客们正在开创着未来，而我们关注真正重要的技术趋势——通过放大那些“细微的信号”来刺激社会对新科技的应用。作为技术社区中活跃的参与者，O’Reilly 的发展充满了对创新的倡导、创造和发扬光大。

O’Reilly 为软件开发人员带来革命性的“动物书”；创建第一个商业网站（GNN）；组织了影响深远的开放源代码峰会，以至于开源软件运动以此命名；创立了 Make 杂志，从而成为 DIY 革命的主要先锋；公司一如既往地通过多种形式缔结信息与人的纽带。O’Reilly 的会议和峰会集聚了众多超级极客和高瞻远瞩的商业领袖，共同描绘出开创新产业的革命性思想。作为技术人士获取信息的选择，O’Reilly 现在还将先锋专家的知识传递给普通的计算机用户。无论是通过书籍出版，在线服务或者面授课程，每一项 O’Reilly 的产品都反映了公司不可动摇的理念——信息是激发创新的力量。

业界评论

“O’Reilly Radar 博客有口皆碑。”

——Wired

“O’Reilly 凭借一系列（真希望当初我也想到了）非凡想法建立了数百万美元的业务。”

——Business 2.0

“O’Reilly Conference 是聚集关键思想领袖的绝对典范。”

——CRN

“一本 O’Reilly 的书就代表一个有用、有前途、需要学习的主题。”

——Irish Times

“Tim 是位特立独行的商人，他不光放眼于最长远、最广阔的视野并且切实地按照 Yogi Berra 的建议去做了：‘如果你在路上遇到岔路口，走小路（岔路）。’回顾过去 Tim 似乎每一次都选择了小路，而且有几次都是一闪即逝的机会，尽管大路也不错。”

——Linux Journal

中文版推荐序

这是一本不错的产品经理或项目经理的入门级手册，对于一些刚选择做产品经理的人来说，能成为指路明灯。我在做了5年技术后转行做产品经理时，也像作者一样，没有人指路，整整自己摸索了一年。如果当初能看到这本书，肯定会少走很多弯路。书中用到了不少作者在谷歌和亚马逊工作时的例子，能帮忙读者更好地理解产品交付的全过程。所以即使是老手，也可以从中收获到一些来自优秀公司内部的最佳实践。

不过在阅读此书时要注意一点，所有的流程和规范最终都是为了解决实际问题的，不要贸然引入流程，除非已经碰到问题，否则尽量简化流程，提高效率。要交付一个产品，其中最重要的只有3点：1）确定用户需求和预期指标；2）以最小成本实现最主要的需求；3）发布并获得数据反馈，确定下一个迭代目标。建议从这个最小集开始，哪个环节出现问题，再参考本书引入相应流程，可能更有效。

另外，附录的十大交付原则中提到“从用户角度出发”，这点无疑是最重要的，这和文中提到的产品定义的第一步撰写新闻稿居然是一脉相承。用用户能理解的语言去描述你要交付的产品带给他们的价值和好处十分重要，如果写不出来，或者写出来用户看了一点兴趣都没有，请尽快停止这个产品，因为即使整个产品开发和交付过程极其完美，这也注定会是一个失败的产品。

最后，请记住：一个互联网产品的交付上线不是意味着结束，而是意味着刚刚开始。所以尽快从产品交付的喜悦中回来，进入下一个迭代，在迭代中和产品共同成长吧！

陶伟华

360手机助手高级总监

前　　言

交付卓越

在软件行业中，我们把设计、打造、发布一款符合市场需求的软件称为交付(shipping)。软件交付可不是打打包，举办个发布仪式那么简单。它要求你找到合适的产品，克服过程中的复杂与多变，并快速完成发布！这个世纪真没生出多少新工种，交付却是其中之一。也许你会质疑，交付不就是“管理”嘛，哪个软件项目没有一个管理者呢。确实，哪里会没有高谈阔论的管理者呢！就算是原始人搬运猛犸骨头，后面都站着一批讨论库存管理的酋长们呢。管理太泛滥了，各种理论层出不穷，有的甚至没有任何实践依据，所以我宁愿使用“交付”这个词。交付这一工作很新，我们呱呱坠地时它还没有出现，甚至等到我们的孩子出生时它也不过刚在角落里长出嫩芽，在学校里更不可能学到它了。

尽管时日尚短，交付却已展现出非凡的价值。它简直是一剂灵丹妙药。它能解决钱的问题，因为投资人给你追加投资的前提是你取得了好结果；它能解决客户的问题，因为交付能力的强弱决定了你是否能推出客户需要的功能和补丁；它能解决团队的问题，因为没有什么比取得进展更能让团队士气大振！所以，如果你想追逐名望、财富、幸福感，那么，交付出卓越的软件，你将赢得一切。

只要精通交付，你就不用担心软件商业化会失败，也不用害怕与大公司展开竞争，因为你的市场嗅觉会更加灵敏，行动更加迅捷！倘若你因不懂交付而导致延期、发布产品时门庭冷落或苦心构建的产品无人问津，你的团队会变得急躁，你的客户会直接写信给你的大老板投诉，而你最好的结局是晋升无望，最坏的则是和你的团队一起卷铺盖走人，你们也终于可以有时间琢磨下简历或者亲自动手洗车了。

所以，精通交付则前途似锦！但要让团队精通交付可不是件简单的事情，不过这

也正是你读此书的原因吧！

这本书将告诉你如何快速精通交付，就好比麦肯锡的“迷你 MBA”培训。这家全球最有名、最昂贵、最顶级的管理咨询公司每年都会招一批科学博士，进行为期两周的称为“迷你 MBA”的培训，培训完成后这些科学博士通常比那些受过两年培训的 MBA 们还要出色。本书将提供给你一个同样简单、精炼的方法，让你轻松完成交付或者更好地理解团队主管的工作。

为什么我想要写这本书呢？当我初入这个领域时，没有前人指路，只能筚路蓝缕，以启山林。后来我发现很多产品经理、测试主管、工程经理以及其他各式各样的团队主管们也像我当初一样迷茫，经受着同样的困扰。但幸运的是，在我苦闷之时我得以与一些“伟大的老师”相伴，如达特茅斯学院、亚马逊公司、谷歌公司，以及我那些错误的商业投资等，我从中学到了很多经验。我想把这些经验总结出来并分享给同行们。

我的第一个老师是我自己开的公司。我那时非常狂妄，以为会写软件就够了，定义最小可行产品、管理项目、迭代、发布、市场推广等其他与软件交付相关的事情都不是问题。我因此得到了很多有价值的教训，也知道了狂妄的代价。我后来又加入了一家创业公司做 CTO 并耗费数年时间在业务扩张上。在这个过程中我得到了一些新的教训并重蹈了狂妄的覆辙。羞愧之下，我只身前往达特茅斯学院的塞耶工程学院和塔克商学院学习了一段时间，并取得了工程管理硕士学位。

离开达特茅斯后，我加入亚马逊担任技术产品开发经理和工程经理（所谓的双比萨团队[①]主管）。在客户评论、个性化、反欺诈基础建设等项目中，我亲眼目睹了杰夫・贝索斯和他的副手们是如何工作的，并试着效仿商界中一些最佳工作方法。

后来我加入谷歌担任高级产品经理。我花了 5 年多的时间研究可扩容性、商业决策以及软件团队内部的人际动力学。我将 Google Pack[②] 发展壮大，把 Google Update 服务应用到数十款产品中，帮助构建采用移动同步服务的 Google Apps 项目、Microsoft Outlook 连接器和数据导入工具。我还推出了 Google 创新多路视频通信产品，

①双比萨团队是亚马逊 CEO 杰夫・贝索斯提出的概念，即团队人员不能太多，两个比萨大家就都能吃饱了。——译者注

②本书中所有的互联网产品名称都将沿用原著中的英文名称，以避免混淆，部分英文名译者会提供注释。Google Pack 指谷歌向用户免费提供的软件包，在第 2 章中会谈及。——译者注

现在它是 Google Hangouts 的一个功能。我甚至为 Google Maps 工作过一段时间。我见证了公司的成长和改变，真切感受到了产品因何成功，却又因何失败。这为我进一步完善软件交付方法提供了大量经验。

关于交付，你可以从亚马逊、谷歌的那些杰出的业务主管那里学到很多。但请切记，交付是个新事物，与此配套的技术、流程、技巧等都极度缺乏。微软倒是有一些软件交付的经验，不过它的方法都是从开发大型、耐用的软件过程中总结出来的，确切来说，就是从开发占据业界统治地位的 Windows 的过程中总结出来的。不过互联网兴起后，那种三年开发周期、通过软盘售卖的老方式已经失去了价值。快速迭代、部署、互联网服务托管已经变成了主流。工程师们越来越关心如何搭建一个可以快速响应的应用开发框架，如何进行可用性的研究，以及如何构建一个更好的 Scrum 这样的过程框架。但是关于交付的知识太少了，亚马逊和谷歌的成功经验在外面流传的也不多，我们更多时候只能摸着石头过河，一不小心就可能误入歧途。

这本书将涵盖我从工作中学到的、提炼出来的关于交付各阶段的较为完备的知识体系。一旦走上了软件交付之路，你将面临产品、方案、项目和工程管理各方面的挑战。因为软件交付不只是如何管理项目，也不只是如何提升开发效率，你必须具备更全面的技能。你既要加深对技术的理解，又要贡献更好的产品创意，更重要的是，整个过程中，你需要展现出你强有力的商业洞察力。你也许要做所有工作，包括要求工程师编写测试用例，或者用 Photoshop 绘制产品原型。这个工作要求你追求极致，只要你不惧挑战，它终将成为你的舞台！

换个角度说，软件交付的过程一定会伴随痛苦、混乱、艰辛。直到游刃有余时，你才能感受到强烈的成就感。这好比在砾石球道上打高尔夫，如果你是新手，一杆挥毕，球不知飞到哪里。球童被你折磨崩溃了，你也不能幸免，整日在烈日下寻找那个正绝望地躺在某个石头底下的小球。但如果你是高手，嘿，连续漂亮的击球后你轻而易举地站到了果岭之上，环顾四周一群新手正汗流浃背地在砾石堆里寻找着他们那个不起眼的球，这一刻你一定知道这意味着什么，这意味着荣耀！

本书将分为两大部分来帮助你精通软件交付。第一部分是关于那些在亚马逊和谷歌做得最好的团队是如何交付软件的。我按照项目开始到发布的顺序来安排章节，包括用户需求研究、用户体验设计、项目管理、测试、发布等。第二部分是关于团队主管带领团队成功交付所需要的技术积累、最佳实践和技能。第一部分建议逐章阅读，

按照软件交付的过程一步步来，第二部分则可按任意顺序阅读。你还可以根据工作需要针对性地阅读某些章节。

本书所提供的工具和建议都是通用的、方向性的。美国西部传奇警长怀亚特·厄普为了能更快开枪，会卸掉柯尔特左轮手枪的安全锁，磨平击锤凸轮。你也可以将这些工具和建议抽丝剥茧以彻底为你所用。如果你想看到对软件策略的深入分析，不好意思，这本书不是你想要的。但如果你想找一个经过实践检验、学习起来简单且能带领你的团队走向成功的方法，那么请继续读下去吧！

鸣谢

特别感谢 Brian Marsh，他是这个世界上最好的工程经理，过去八年我和他在同一间办公室里共事，他帮我弄明白了什么是交付。这本书中有很多内容来自于他的建议（当然不是那些蹩脚的笑话）。非常感谢 Aaron Abrams，他是我最好的读者，他第一个告诉我要“让观点更犀利点儿”。谢谢 Ali Pasha、Steve Saito、Matt Shobe 和 Mike Smith，你们阅读了初稿并提供了非常棒的反馈。最后，特别感谢你，Tim，感谢你的耐心、无微不至的帮助和永远的支持。

目 录 CONTENTS

第一部分

交付卓越产品，步步为“赢”

SHIPPING

GREATNESS

打造一款卓越的软件似乎并非难事，但事实往往不尽如人意。通常产品要么逾期交付，要么没抓住真正的用户需求，要么发布后还有一个又一个烂摊子等着去收拾。究其原因，很重要的一点便是我们不知道如何正确地组织交付过程。我们经常会舍本逐末，将关键步骤忘个干净，或者瞎冲乱撞，一头扎进琐碎之中出不来，以为凭借运气、加班加点和美好的愿望就能把产品成功推向市场。

这样做也许能成功一次两次，但却不可持续、没有效率，所以亚马逊的顶尖团队不会这么做；这样做也毫无乐趣可言，所以谷歌的顶尖团队未予采纳。其实有一套更为有效的交付过程，它分成 7 个阶段，任何团队主管都可按图索骥，收获成功与快乐。

阶段一，确定正确的产品方向。如果总是设计垃圾软件，那么你永远不可能交付卓越的产品。好的产品一定要满足众多客户所共有的某个真实的需求。你的使命就是找到一种独特而有意义的方法去满足这一需求，并且在交付过程中的所有努力都应围绕着这个使命。例如，你应根据使命来制定产品切入市场的策略。一旦确定了使命和策略，产品定义就会更清晰，而不会沦为垃圾，因为它完全符合你出色的策略。

阶段二，尽可能清晰详细地定义产品。这个过程需要 10 个主要步骤，包括撰写新闻稿、创建并不断更新 FAQ 文档、撰写功能需求文档等。完成这些步骤后，工程团队就会对项目形成统一的认识，管理层或投资者也会了解并认可产品的设计。这时候所有人都会异常兴奋，而你也可以稍作休息了。

阶段三，设计用户体验。你需要从用户的角度出发，和设计团队不断沟通、反复迭代，最终构建出良好、直观、简洁的用户体验。你应不断提出问题，促使设计团队始终围绕着产品使命展开工作。你还应该让工程团队和设计团队保持密切合作，确保设计最终可被实现。

阶段四，做一些基础的项目管理工作，不要太多也不要太少。当工程团队拿到详细的产品原型图和需求文档开始编码后，你就需要做一些基础的项目管理工作了，包括跟踪交付物的进展、指出问题以及控制项目范围。

阶段五，开始测试。随着各个功能的代码块陆续提交，实际产品逐渐浮出水面，团队的工作速度将会提高，测试团队也将开始全心投入测试。这一阶段虽不需要多少创造性，却依旧令人兴奋不已。作为团队主管，你需要主导 Bug 的处理并慎重决定哪

些可以容忍出现在版本 1 而哪些又必须在发布之前修复掉。

阶段六，你差不多可以准备发布了，不过在发布之前要清楚知道怎样才算成功，这就要求你建立一套衡量产品成败的指标。让团队利用剩余工时来把这些指标纳入监控并搭建产品状态面板。当产品 Bug 趋近于零时你就可以准备监控产品发布效果了。记得买好香槟放在冰箱里以备庆功时尽情畅饮。

最后，正式发布产品。发布一款卓越的产品可不只是上传一些文件到服务器上那么简单，你需要制订市场营销和公关方案，并在发布前仔细核查清单中的每一项内容。基本上每次发布都会有一些糟糕的事情发生，不过只要处理得好，大部分用户都不会察觉到。记得观察产品状况面板各项指标的走势，它会告诉你是否正走向成功。

纵观以上 7 个阶段，推出一款卓越的软件并不是件难事。你只需按顺序完成各阶段的具体任务，就能打造快乐的团队，交付成功的产品。

从以上阶段可见，我们一直在努力缩小项目范围，简化用户体验，提升推进速度。整个过程越快，迭代就越快，交付的产品也就越好，因为每次迭代都会吸收上一次迭代中客户提出的建议和意见。虽然每个产品有每个产品的功能，但它们的交付流程是一样的，所以你只要熟练掌握上述 7 个阶段，就能成功交付所有产品。下面让我们来详细考察每个阶段，先从制订使命和策略开始。

第 1 章 赢在使命和策略

1

SHIPPING GREATNESS

除了获得财富和名望，成功交付的关键还在于快速且充分地满足客户需求。因此，你的使命就是解决客户的问题，你的策略就是找到一种独特的方法来满足这个群体或细分市场的共同需求。这听起来简单，理论上也不难。但就像赛车，理论上你只需在正确的位置刹车、转弯和加速就能赢得比赛，但实际上你很难做到这一点，让车子在跑道上发挥出最佳性能并不简单。同理，要想准确识别客户需求并围绕客户需求构建使命、制订策略也实属不易。为此你需要掌握一些特殊的技能，并对一些要点给予特别的关注。下面我将一一道来。

首先来看看如何找到一个大众共有的需求。

1.1 如何找到正确的需求

如果仅凭别人一句“哇，太酷了，就做它吧”就确定要开发某个产品，财富、名望和成功绝不会向你招手。你可能想迎合某类群体的需求，可是他们有很多不同的需求，你又怎么确定首先要满足哪个关键需求呢？还是让我们回过头来，先看看那些极为富有的著名成功人士有什么好建议吧。

亚马逊 CEO 杰夫 · 贝索斯一直强调团队必须“以客户为导向，而不是以竞争为导向”，公司业绩因此持续攀升，股东也获益不小。他的观点非常清晰地揭示出这样的原则：团队应该始终积极地去解决客户的问题，而不是紧盯竞争对手，被动地做出反应。无独有偶，谷歌 CEO 拉里 · 佩奇也常说一句类似的话：“立足客户，向外拓展。”他的理念与杰夫相似，只是更为抽象一些。从拉里和杰夫身上，我们学到必须专注于解决真正的客户问题。

谷歌创始人兼总裁谢尔盖 · 布林对此也有些真知灼见。他多次提及：“不要只想着解决简单的问题，越困难的问题越值得去努力。”当把一个问题不断放大时，你覆盖的客户会不断增多，而问题的解决也会使更多人受益，这意味着你的潜在收益会更大，财富、名望、成功也就随之而来了。如果说拉里和杰夫告诉你必须解决真正的客户问题，布林会告诉你这还不够，你得确保这是个很多客户都存在的大众问题。

我们以 Google Pack 为例来看看谷歌是如何解决一个真正的难题的。Google Pack 是一个免费软件集合，为个人电脑提供实用软件，我曾在 2007 年 ~2008 年负责这个产品。当时我们发现很少有用户去升级他们安装过的软件，因为升级过程极为复杂。而用户不升级软件，系统速度就会变慢，安全漏洞就会增多，孩子们在假期使用电脑时也会遇到很多麻烦。

我们提出的解决方案是不设法优化每一个关于升级的用户体验，而是构建一套无需用户操作即可自动升级所有软件的系统，而且这套系统对第三方软件同样适用。显然这个方案更难实施，特别是考虑到第三方软件各有各的升级安装过程，但一旦构建成功，就能帮助到数以亿计的用户。Google Toolbar 和 Google Chrome 后来也使用了这套系统，最终它以 Google Update 开源软件的形式推出供所有人使用。我认为这是一个极其成功的产品，它作为平台被业界广泛应用。而它之所以能够成功，要诀便在于它解决的问题比我们最初意识到的那个问题要难得多！

如果你已经找到了这样一个大问题，那么就完成了产品定义过程中最重要的一步，而且更为重要的是你将开始以一种富有意义的方式去帮助一大批人！真实、大众、共有这些大问题的标准尽管很明显，却依然经常被忽略。这些标准还构成了使命的基石。下一步你应围绕这些基石构建使命并制订策略。

1.2 如何构建卓越的使命

团队一定要有自己的使命。如果你没有清晰地阐述使命则会导致失败，因为你的团队、组织和投资人会根据自己对使命的不同理解而各自为战，从而导致冲突、混乱和痛苦。这种情况屡见不鲜。还有些团队不愿意清晰地阐述使命，他们害怕会因此陷入到使命是什么的争论中去，但这不过是掩耳盗铃，一旦人人都意识到自己和别人步调不一致，争论将不可避免地爆发。因此，只有尽早确立卓越的使命，才能真正从总体上减少冲突，解决这个问题。

卓越的使命需要完全符合以下三点要求——只有这三点。

1. 能够唤起人们的兴趣

要想吸引人并使他们认同你的使命，最重要的是确保你的使命能够唤起他们的兴趣。只有长期吸引利益相关者的注意，你才有时间去挖掘产品细节。

2. 提供言之有物且能指明方向的原则

你的使命应该能够指导你朝着哪个方向去努力。千万不要把使命定成“永争第一”之类小学生常喊的口号。通过在使命中指明方向，你将希望达成的结果清晰化了。

3. 适合印在T恤上

你也许不打算把使命印在T恤上，但不妨试试看，人们会更容易记住它。而只有团队牢牢记住它，他们才能依照使命做出各种决定。不要以为有一个高智商天才组成的团队就不必这样做，他们的天才能力可不一定表现在对这类事情的记忆上。因此，为了让你和你的团队更容易记住使命，应控制它的长度，适合印在T恤上就对了。

我们举个例子来说明什么样的使命能够满足这些要求。亚马逊有一个非常出色的主要负责产品个性化推荐的团队，他们为自己定义的使命是“带给客户更多欣喜”。这个响亮的使命完全符合以上这些要求。

- 能唤起人们的兴趣。因为每一位团队成员都希望自己的工作能带给客户欣喜。
- 言之有物且指明方向。它指明了我们的方向是开发更多让客户欣喜的功能。它还言之有物，让人想到了意料之外的发现、探索以及愉悦感。
- 适合印在T恤上。虽然过去这么多年我还依然记得它。

最后一个衷告：你需要的是一个能够反映代表性产品或服务的使命，而不是一个面面俱到的使命。

1.3 如何制订正确的策略

很多人觉得制订策略是一件高深莫测的事，这可能是由很多方面的原因造成的。比如业界的咨询公司为了赚取更多的钱，会把这一过程人为地复杂化。当工程经理面对问题感到束手无策时，就会炮制出一些旁人看来难以理解、似是而非的策略来。但一个更为常见的原因是策略本身就是个模糊的东西，有时候你明明心中有一个很好的想法却不知如何表达出来。不过幸好我们是在软件行业，可以大刀阔斧地简化策略制订过程。

首先我们要知道什么是策略。策略是指在竞争对手的压力下，利用公司独特的优势来争取目标用户的粗略计划。它就是这个样子，既不是详细的产品描述，也不是一

页细致入微的计划。它只是一段用于说明对目标客户来说你的产品将如何长期保持比竞争对手更强的吸引力的话。简而言之，你需要阐明三件事：客户、公司和竞争。

举个例子。我在 Google Talk 时肩负的使命是“使人与人之间在任何地点均能通过任何终端沟通”。纵观统一通信、视频会议和 VoIP 领域的竞争格局，我发现谷歌有一个竞争对手短期无法赶上的独特优势，即它拥有庞大的云服务基础设施，我们可以在此基础上利用交换技术来提供视频会议服务，这比 Skype 或其他视频服务供应商过时又昂贵的混合编解码技术要先进得多。部署一套他们的多通道视频系统通常需要上万美元，并且由于硬件原因，使用过程中会有很多延迟，实际效果不尽如人意。谷歌则没有这个问题，它的技术独一无二且竞争对手短期内难以复制，因为这个技术需要一个巨大的数据中心，而没有人能建立比谷歌更庞大的数据中心。

因此无论是从公司角度还是从竞争对手的角度来看，这都是个合适的策略，我们能够通过这种独一无二、成本低廉的服务来取得领先地位。通过观察 Google Apps 数百万的客户与行业的发展趋势，我发现了一个新兴的，是由两类客户组成的细分市场，其中一类是企业中在不同地域工作的员工，另一类是在家办公的人，这两类群体都在日益扩大。此外电话会议市场的发展潜力也很大，而针对这个市场我们有优势明显的 Google Voice 服务。

综上所述，我认为可以通过为企业提供低成本的统一通信服务来赢得市场。这个策略能让我们为中小企业以及中端市场提供比 Skype 更先进、比微软更低廉的服务。虽然最终谷歌没有采纳它，而是将 Google Talk 和 Google Voice 应用在 Google+ Hangouts 上，重点发挥它们的社交属性，但你应该能从上面的思考过程中了解到该如何制订策略！

当你开始思考公司、客户和竞争这三大问题时，需特别注意如何才能长期为客户提供比竞争对手更优质的产品。这也是在交付过程中唯一一次需要考虑竞争的时候，所以一定要想清楚！你需要深思远虑，因为要想取得商业上的成功就必须保持长期的竞争优势，否则竞争对手就会迅速模仿并推出一个和你的产品功能一样、价格却更低廉的新品牌来将你一举击溃。

既然现在你已经知道了谁是产品的忠实拥趸以及产品如何保持长期的竞争优势，那么请用最多三段文字写下来，然后再将这些想法的本质浓缩成一段文字。越简短的策略越容易实现，也越容易获得他人的认同。

这里再举一个例子。假设我们是亚马逊旗下子公司IMDb（Internet Movie Database）的产品负责人，经过一轮头脑风暴后制订了如下使命。

使命：启发视频观看者。

这个使命能唤起人们的兴趣吗？我认为能。“启发”这个词就引起了我的兴趣。可能对于一些工程师读者来说更是如此，不过先看看我们的产品是如何适用这个词的吧。我们把“启发”定义为提供背景资料、鼓励探索甚至是帮助你了解朋友们的想法等。所以看起来是合适的。

这个使命是否言之有物并指明方向？当然。它告诉了我们需要关注的对象，即观看者。这里，我没有限定“观看者”必须是“电影观看者”，因为我认为YouTube、Hulu等平台的观看者也是我们的目标群体，但不包括照片或其他形式艺术作品的受众，所以我用“观看者”这个词来限定需要关注的对象。同时“启发”这个词告诉我们应提供信息和数据服务，因此这个使命指明了团队应为目标用户提供哪些服务。

这个使命适合印在T恤上吗？很适合，只要别把它翻译成德文并用大号字体印刷。

现在我们有了一个大家都认同的使命，接下来便围绕这个使命制订策略。

策略：随着越来越多内容的产生，用户每天消费的内容也越来越多，但对于20～40岁的工薪一族来说，他们在海量的内容面前却不知道如何选择。我们需要给这些用户以启发，帮助他们找到想看的视频，并让他们在看的过程中对内容有更深的理解。

我们之所以首先选择工薪一族是因为，与有大把的时间耗在Facebook和YouTube上的青年不同，工薪一族时间有限，但他们人际网络丰富、个人主见强烈，还有可自由支配的收入，愿意在内容上消费。

我们有独特的方法来解决用户挑选视频的问题。通过整合IMDb独有的电影数据集、亚马逊对数码内容分门别类的能力以及可靠的个性化推荐技术，我们可以构建出有效的视频推荐算法。虽然其他竞争对手（如Netflix）也有一套推荐引擎，但我们覆盖的平台多，拥有的数据也更丰富，能够提供比竞争对手更有趣的观看体验和更精准的推荐。

我们将把这种观看体验通过各种载体来传递给观看者，这些载体展现了与内容相关的背景数据（如YouTube视频的演员阵容），包括YouTube等网站的浏览器插件、

手机应用程序等。我们还会提供丰富的与内容相关的信息以启发观看者，并提示观看者进行反馈——这就创建了一个良性循环并惠及所有用户。

这个策略满足了所有要求：阐述了 IMDb 应该提供什么产品以及为什么这家公司适合提供这类产品，分析了竞争对手的情况以及 IMDb 应如何与之差异化竞争，还论证了 IMDb 为什么要针对这样一个特别的消费者群体。该策略不但简明扼要、详略得当，还直接指明了我们的具体目标，即整合所有视频来源并展现影片背后有趣的故事。

你是否注意到一个小细节？在我写目标要瞄准工薪一族时用了“首先”这个词。它的言外之意是“青少年的需求将在版本 2 中考虑”，虽然这有点像空头支票，但好在能让我们先专注于一个更小的群体且无需彻底否认青少年的重要性。12.2 节介绍了更多有关“放到版本 2”技巧的信息。

写完一个基本成型的使命和策略后（你可能已经写得过于具体了），你该坐下来和主管讨论一下你所写的东西。这是获得大家认同与支持的第一步。如果在这一层面都无法取得共识，你就寸步难行了，因为后续要做的只不过是不断细化使命和策略而已。

当你和团队主管们取得初步一致后便可以定义产品细节了。

第 2 章 赢在产品定义 2

SHIPPING GREATNESS

交付的下一阶段是让你的产品方案具体且可理解。通过制订使命和策略，你知道了你的客户是谁，他们的需求是什么。你也知道如何做才能比对手更出色、更具备差异性。有了这些知识再加上一些头脑风暴，便可以得出一个大致的产品方案。或者你像我们大多数人一样，老板会对你说：“那就做出来看看吧。”这时需要借助文字来告诉你的团队你要做什么事情以及目标是什么。换句话说，需要思考用什么样的文字才能非常贴切地描述出你的产品，使设计师可以借此设计原型，HR 可以借此雇佣工程师，而你可以借此拿到项目经费去购买面包和服务器！

当设法细化产品方案时，你会发现产品要解决的一些客户问题都是你主观臆断的，而且因为你的使命和策略都是建立在客户问题上的，因此这些主观臆断也混入了其中。我不想扫你的兴，但事实是你的这些臆断很可能是错的。这一点儿也不奇怪，亚马逊、谷歌和其他大公司都犯过这样的错误。所以要采用一些方法来证明臆断是否正确。即便它们十有八九是正确的，也要经过证明，而证明的最好方法就是把产品提供给客户使用，然后听听他们的意见。

多次在软件行业创业的埃里克·莱斯比较认同这个方法。他在《精益创业》一书中充分论证了为什么该构建一个*最小化可行产品*。最小化可行产品是指产品最小的组成部分。通过把它提供给一定量的用户使用，你可以验证之前关于客户问题的臆断是否正确。你可以视需要来定义最小化可行产品、选择参与测试的客户数量以及决定一次验证几个问题。但不管最小化可行产品是大是小，你都可以参照下面的产品定义过程来定义产品。通过快速重复这个过程，你可以验证不同的客户问题，并添加更多更好的产品特性。当迭代越小越快时，你甚至不需要花大力气去猜测客户的需求，而是更多按照客户告诉你的去做，这样成功的可能性更大。

产品定义过程主要分为 10 步。每一步都建立在上一步完成的基础上。其中有些步骤比较简单，有些步骤（比如撰写新闻稿）你可以只在首次迭代时进行（当后续的迭代都只是小的功能升级时），所以整体上来看产品定义过程没有想象中的那么长。当你确定了产品策略后便可以开始产品定义的第一个步骤了。等到 10 步都完成后，你便拥有了一份定义完整、描述清晰的产品文档，你的工程团队也可以进入编码阶段了。

1. 撰写新闻稿。亚马逊喜欢这个不同寻常的第 1 步。新闻稿是一篇脱胎于策略的

文章，篇幅不超过1页，主要用于促进各方了解和公开透明。你可能需要几天时间才能成稿。

2. 创建并不断更新FAQ文档。这份“活跃”的FAQ主要用于记录一些争议点和重要细节。你可以花1个小时搭建框架，然后在开发过程中以及上线后抽一些“业余”时间更新维护。你可以使用Wiki或Google Doc等现成的工具搭建和维护FAQ，这样可以节省费用。

3. 绘制线框图或流程图。线框图和流程图是产品的可视化描述，在讨论或答疑中使用可以让观点更明晰。绘制可能需要1天到1周不等的时间，鉴于这是最有力的沟通工具，还是值得投入这么多时间的。

4. 撰写产品单页或10分钟的演示文稿。产品单页是一篇写给高管或多数风险投资人看的产品介绍文章，你需要把控好介绍的详略程度。演示文稿和产品单页内容一致，只是表现形式不同。你可以花几个小时的时间写份初稿，然后收集一些同事的反馈并做修改，如此反复几次便可定稿，整个过程通常需要1～2周。

5. 在FAQ中添加API文档。API是产品的第一个技术触角，在第6步会把它们全部整合到需求文档中。你可以先花数小时简单起草一些API，后续再在工程团队的帮助下完善它们。

6. 撰写功能规格文档。功能规格文档又名产品需求文档（Product Requirements Document，PRD，谷歌常用此名），或市场需求文档（Marketing Requirements Document，MRD，微软常用此名）。它是阐述产品各功能详情以及为什么要有这些功能的最权威的文档。你可以将新闻稿、FAQ、线框图、产品单页和API文档等内容粘贴到功能规格文档的不同章节中。除去这些主料，还需要添加负载规划、非目标以及非常见用例（用于清晰表述FAQ中各种非常见情况的用例）等佐料。

产品的规模以及成熟度决定了你需要几天还是几周才能写完功能规格文档。如果产品尚不成熟，你应当尽可能缩小产品规模以快速验证客户需求的真实性。如果产品规模庞大且已臻成熟（如苹果的iPhone），你就需要一份健全的功能规格文档了。

7. 邀请设计团队和工程团队主管参与产品评审。这一步是为了获得项目组对产品的认可，并让他们帮忙寻找产品中存在的各种极端及边缘情况。你应该邀请他们一起评审产品，这样一天时间评审就可以完成。评审过程中他们可能没有贡献细致的反馈，但至少有机会去想一想你的产品方案，否则等到猴年马月他们也未必会读你的文

档。至于如何让你的团队参与阅读和评审，你应该已经驾轻就熟了。

8. 找客户测试产品概念。在此阶段，你需要了解产品方案是否真正能解决客户问题。花一天时间做一次完整的认知走查①或花数天时间进行在线调研都是不错的测试方法。

9. 命名、定价以及预测收益。你可以晚些时候再想这些事情，只要对产品有足够的信心。对我而言，这一步的价值是让我睡得更为安稳，因为通过收益预测我会更加确信我的产品能给投资人带来回报。但我并不认同一些 MBA 花两周时间去弄这些东西，我觉得你只需（而且应该）投入几个小时或者更短时间。如果花的时间比这长，那你肯定是想多了。

10. 向管理层汇报。上面 9 步完成之后你便可以向管理层或 VC 汇报你的产品了。你可以使用 10 分钟产品演示文稿来汇报，并视需要展示 FAQ、线框图等。一旦通过，就可以动工了。通常汇报需要 30 分钟或者更长。

你是否觉得这样的产品定义过程太难了？不用担心，你的能力足以应付！这个详尽的产品定义过程已经把七零八落的事项重组成了一串非常细致但又连贯的工作。只要循序渐进，为跨越每一步而欢欣鼓舞，你就会拥有一段快乐的时光。产品定义过程的大多数步骤(第 6 步和第 7 步除外)都很快而且有意思(如果你和我一样也是个极客)。让我们从第一步开始吧。

2.1 第 1 步：撰写新闻稿

撰写新闻稿是产品定义的一个另类却有效的开始。它是由亚马逊 CEO 杰夫 · 贝索斯率先倡导的。所谓新闻稿是指一篇向市场宣布将要推出新产品的通告。不管是新闻稿还是博客文章，都应该简单明了地传达关于产品的关键信息。之所以选择撰写新闻稿作为第一步，是因为相比 FAQ、产品单页而言，新闻稿的媒体属性注定了它天生就更简洁、可读性更强且更关注真实的产品能给真实的用户带来什么价值。

①认知走查是指通过分析用户的心理加工过程来评价用户界面的一种方法，最适用于界面设计的初期。分析者首先选择典型的界面任务，并为每一任务确定一个或多个争取的操作序列，然后走查用户在完成任务的过程中在什么方面出现问题并提供解释。本解释摘自周荣刚和张侃所著的《界面可用性评价之认知过程走查法》一文。——译者注

好的新闻稿包含以下六大要素：

- ■产品命名
- ■发布时间
- ■目标客户
- ■解决了什么问题
- ■如何解决（务必简明扼要）
- ■CEO 的公开赞辞

请注意，新闻稿不要深入细节。它很少包含图片且从不包含财务信息。它只是从用户视角出发简明扼要地阐述产品是什么、什么时候发布以及为什么要做这个产品。如果你已经确立使命和策略，那你应该思考过如何从用户视角来阐述产品了，这会让你在撰写新闻稿时得心应手。事实上新闻稿的这些要素都是直接来源于你的策略，比如 CEO 所表扬的内容正是你那套切入市场的独特方式。

我在负责 Google Apps 时曾帮助撰写了后面这篇博文，虽然只在前面几段序言中探讨了大致的业务难点（在 2009 年企业用户还难以部署 Google Apps），但大体上你还是能看出一篇好的博文是什么样的。因为只是一篇博文，所以没有引用 CEO 的赞辞，而是换成了一个大客户的推荐语。这篇博文的效果令人兴奋不已，它让我们知道了这个产品是如完美地契合了客户的需求，而这不正是本阶段我们要达成的目的吗？（访问 http://googleenterprise.blogspot.com/2009/06/use-microsoft-outlook-with-google-apps.html 可查看完整的文章。）

使用 Microsoft Outlook 管理 Google Apps 的邮件、通讯录及日历

2009 年 6 月 9 日，周二

去年一年我们都专注于如何降低企业部署 Google Apps 的成本。在过去的几个月已经有若干改进面世，包括离线 Gmail、用户文件夹同步以及与黑莓的全面互通性。

今天我们非常兴奋地宣布又一项改进面世了，它就是面向 Microsoft Outlook 的 Google Apps 同步插件（Google Apps Sync）。该插件允许你通过 Microsoft Outlook 无缝管理 Google Apps 专业版或教育版，从此企业部署 Google Apps 的又一个关键障碍被排除了。

相较于商务用户以往使用的产品而言，Gmail 的界面和特性更受他们喜爱，但有些时候总有一批用户只喜欢 Outlook。为了方便他们使用 Google Apps，我们开发了

面向 Microsoft Outlook 的 Google Apps 同步插件。它允许 Outlook 的用户管理 Google Apps 的商业邮件、通讯录和日历，并且当他们的工作电脑不在身边时还可以通过 Gmail 的网页端来获取这些信息。

它的核心功能点如下。

- 邮件、日历和通讯录同步。同步插件使用离线 Gmail 协议来同步邮件，比 IMAP 及其他协议速度要快很多。
- 空闲 / 忙碌状态查看和全局邮件列表功能，不管你的同事是使用 Outlook 日历还是 Google 日历，你都可以轻松发送会议邀请。
- 简单的数据迁移工具。只需要两次点击，企业雇员就可以把 Exchange 或者 Outlook 的现有数据复制到 Google Apps 中。

如果读完之后产生这样的感慨：“如果我是谷歌 CEO 埃里克·施密特，我肯定会明白为什么这些家伙愿意为这个项目付出数年努力，这绝对值得！”那么你已经明白了新闻稿的作用，马上开始写你的产品的新闻稿吧。但如果你心里觉得：“这家伙就是一个脑残，什么时候我们才能开始写 API 啊？”那么没有关系，跳过这一步吧，但请不要再跳过其他步骤！

如果跳过了撰写新闻稿这步，等到完成后面两步时，你会发现仍然需要写一篇与新闻稿类似的文章，即产品单页。由于没有写作新闻稿这类面向客户的文章的经验，你会发现产品单页难以下笔，所以最好不要跳过第一步。但在某些情况下跳过这步又是合理的，比如是内部系统开发、产品特性改进或者处于一个新奇事物（如产品开发前的新闻稿）不受欢迎的公司环境中。

2.2 第 2 步：创建并不断更新 FAQ 文档

随着产品方案的不断细化，各种问题也层出不穷，其中大部分都非常重要，指出了产品的不足之处。我会迅速把这些问题记到一个内部 FAQ 文档中并尽我所能回答提问者。我喜欢那些愚蠢的问题，因为它们让我感觉好像没花多少精力就消灭了一个问题，这真是一种少有的乐趣啊。

如果觉得某个问题外部用户也会问到，我会把它记到 FAQ 文档的“外部问题”部分。通过不断添加新问题，这份文档将变成那些有疑惑的人寻找答案的动态更新的

可信资源。

当遇到回答不上的问题时，我也会把它放入 FAQ 并期望有人能够回答它。最坏情况下你也可以把这个 FAQ 当做个人的 Bug 列表或者团队讨论主题库，当悬而未决的问题趋近于零时，你就能够写出出色的产品单页或产品需求文档了。

创建并维护 FAQ 文档有两大好处。第一，它能节省你大量回复邮件的时间，还能抵御一些内部责难。节省时间是因为 FAQ 已经回答了那些显而易见的问题，你就不再需要对每个问题都回复一封长长的邮件。能抵御内部责难（这点更为重要）是因为你很可能因为一些小问题上的疏忽而被问责，这时你可凭借细致周到的 FAQ 文档来证明你是尽职的，它会帮你浇灭大部分无谓的怒火。

第二，当你的客户支持团队和科技写作团队开始整理所有面向公众的内容时，FAQ 将是一个很有价值的资源。由于你已经把 FAQ 分为了内部问题和外部问题两个部分，因此他们知道哪些内容可以公开。所以创建并维护 FAQ 文档是个非常出色的主意，在产品研发过程的诸多关键时候它都能帮你节省时间，比如在产品发布前你们团队还需要依靠它来完善帮助内容。

2.3　第 3 步：绘制线框图和流程图

在 FAQ 中撰写问题答案时，你会发现其中一些答案用流程图或线框图来表述会更好一些，尤其是涉及用户体验（UX，User Experience）的细节时。确实如此，流程图可以帮助你准确地解释用户工作流和系统交互相关问题，简要线框图则可以帮助你具象化产品各环节的用户体验，并在之后的汇报中发挥不可思议的作用。除此之外，在白板或空白纸上画草图并用手机拍照也是一个沟通想法的好办法。

因为绘制线框图或流程图非常重要，我会在第 4 章用一个完整小节来讨论绘制的过程和方法，这里不再赘述。

2.4　第 4 步：撰写产品单页和制作 10 分钟的演示文稿

到现在为止，关于客户是谁、要解决他们的什么问题以及采取哪种解决方案应该都有定论了。接下来你需要去争取工程团队、管理层、VC（风险投资人）或其他利益

相关方的初步支持。你需要弄清楚他们对产品的认可程度，否则等到第 7 步功能规格文档都快完成了，而他们还对产品的价值存有疑义，你将面临不断的返工。另外在这时准备演示还有一个优势，因为经历上面几步后你对产品方案的细节以及可能受到的挑战都有了更全面的理解，所以在应对他们的诘问时能胸有成竹、进退有度。

这一步你只需准备产品单页和 10 分钟的演示文稿，也可以两者选其一。在产品单页中根据需要来添加线框图或流程图就可以了，不必考虑它们所占用的篇幅。

在亚马逊，产品单页尤为重要。高级副总裁们（SVP，又称 S 团队）会围坐一圈安静地阅读你的单页，然后讨论是否通过。整个现场就像高考一样，身边每个人都想第一个完成考卷然后逃离这个充满诡异气氛的考场。但不管怎么样，这样的决策机制已经运作数年了。

谷歌的决策机制与亚马逊不同。在谷歌，即使准备脱离幻灯片直接介绍产品，你也需要准备一份演示文稿，因为需要你亲自演示，且谷歌没有建立像亚马逊那样的“高考”机制。第 5 章会谈到如何制作卓越的 10 分钟演示文稿。

面对 VC 时，两样都需要准备，因为你既需要发邮件介绍你的产品，又需要面对面做产品演示。不过无论你在哪里工作，这两份文档的内容都是一样的，它们都是新闻稿的延伸。下面介绍这两份文档所需包含的五个要素：

- 产品名称
- 目标客户 + 数量有多少
- 解决了什么问题 + 这个问题对于目标客户来说有多大价值
- 解决方案 + 这个解决方案类似线上哪个产品，为什么你的方案能让竞争对手在长时间内都无法模仿
- 何时交付 + 主要的里程碑有哪些？
- 团队背景（仅针对 VC）

你会发现产品单页和演示文稿实际上是新闻稿的延伸，它们增加了市场机会（用户量）、收益机会（解决方案的价值）和长期竞争优势（对手长时间内无法模仿）这三方面内容。如果你还不能在产品单页中清晰地表述以上几点，请继续努力！

在 10 分钟的演示文稿或产品单页中同时包含这 5 个要素似乎是件充满挑战的事情，大多数团队负责人初次尝试都无法做到既完备又简洁。我见过成百上千寻求投资或收购而做的产品演示，其中只有极少数能够快速、清晰、连贯地把产品介绍到位。

因此，每个团队负责人都应把进行一份简洁、清晰的产品演示视作一个必须完成的任务，它能帮助你立即赢得尊重、关注并拿到产品启动的通行证。第 10 章有关于如何做好演示的详细介绍。

最后再提一个关于 10 分钟产品演示的建议：你的听众无论来自公司内部还是公司外部，无论关心技术还是关心商业，他们大部分人对你的行业都有充分的认识和独到的见解，但并不了解你要做的事情的前因后果。因此你演示的最佳方式是先讲用户现状，再延展开来（参考先前的五大要素），迅速把要点讲完后再留出时间供这些聪明的听众就他们关心的点刨根问底。不要埋怨他们咄咄逼人，他们并非喜欢看你理屈词穷的样子，这只是他们考察你的一种方式，所以欣然面对吧！

2.5　第 5 步：在 FAQ 中增加 API 文档

API 文档可以说明你的团队如何与其他团队协作、外部开发者如何使用这套系统以及你需要存储什么数据。预先定义清楚 API 还有个好处，它可以帮助你搭建由这些 API 构成的面向服务的体系架构（SOA，Service-Oriented Architectures，第 3 章有更详细说明）。因此预先撰写 API 文档对每个人都有很大帮助。

不只如此，API 最重要的作用是明确系统的各个边界，而明确的边界有助于人们了解各类功能或输出分归哪方负责。这样在项目初期各方就建立了理解和术语上的一致性，为后续高效沟通产品需求打下了坚实基础。

虽然 API 的作用很大，但也会带来一些麻烦。和你合作的开发工程师或许会觉得你抢了他们的工作，所以谨慎起见你应先了解他们的立场。如果理解你撰写 API 的目的是为了让高层同意你定义的关于数据存储和接口维护的责任归属，他们很可能会默许你的做法。如果工程师们依然不理解你的良苦用心，不要继续纠结，换个方式来说服他们吧。记得提醒自己工程团队是你的左膀右臂，别让任何事情影响到你们的关系。

举一个提前撰写 API 的例子吧。当时我们在亚马逊负责构建客户评论中的内容评分系统。作为产品定义的一部分，我需要告诉客户评论团队如何调用我们的系统拿到评分，于是我在 FAQ 中增加了一个简单的 API ：

```
Float getContentQualityScore(string reviewId,
    string userId){}
```

这个API(用某种未知的程序语言写的)相当简单，但依然说明了一些重要的事情。

- 我们假定了索引值不是ASIN（亚马逊产品ID），而是评论ID。
- 我们假定了评分为非整数数字。
- 我们未来想支持类似Netflix的评论评价系统[①]，这样就能给你这样的用户提供最有价值的评论了。所以我们认为API最好也提供评论的评价数据。

你可能想更简单一些，但对于提供给开发者的东西，即使他隶属于其他团队，你都应该考虑得更周全一些，以防止后续出现问题。比如在上面这个例子中我们若不说明评分是一个非整数数字，客户评论团队就会以为我们提供的是类似ABC这样的等级评价，而按等级的排序算法和按分数的排序算法是截然不同的。

2.6　第6步：撰写功能规格文档

你已经精心构思了一个可以解决真实客户的真实需求的产品方案。你也完成了演示并赢得了重要利益相关者的认可，然后又与依赖方一起定义了系统该如何交互，现在可以考虑实施细节并制作一份详尽的功能规格文档了。

微软称这份文档为市场需求文档，因为它整合了市场研究者从客户访谈中收集到的一系列需求。谷歌称之为产品需求文档，因为它通常是由产品经理负责撰写的。亚马逊则把它称为功能规格，因为它描述的是产品应提供什么功能给用户使用。虽然各家命名不同，但对于它功用的看法却是一致的：它是用来详细描述用户应该如何体验产品的文档。它不包含系统在后台如何运行之类的技术细节，这类细节应该包含在工程主管撰写的技术规格或设计文档中。

功能规格文档的读者一般为你的工程团队、设计团队，偶尔还有市场营销团队。功能规格文档包含以下九个内容块，按照先整体后细节的顺序排列，我将逐一详细介绍：

1. 简介（使命和策略）
2. 目标与非目标
3. 用例或用户场景

①用户可以通过点击有用或者没用来评价其他用户撰写的评论，豆瓣也有该功能。——译者注

4. 原型图或线框图
5. API
6. 负载规划
7. 依赖
8. FAQ 和开放问题
9. 关键事件

2.6.1　简介

这块内容与产品单页相同。你也许认为没有必要把它放在这个文档中，但你的工程团队需要它。它说明了为什么要做这个产品以及做些什么，每个新进入项目的成员都可以从中了解到必要的背景信息。同时它还说明了文档中一些术语的含义，你可能因为使用习惯了这些术语而忘记别人其实并不理解。

2.6.2　目标与非目标

简介中虽已大致描述了产品的方向，但你需要将其细化成不同目标，每个目标都应保持清晰简洁并将它们按优先级排列，这样工程团队就可以合理地进行设计与开发了。

如果设定的某个目标表面上与产品方向没有多大关联，你需要解释清楚为什么将它设为目标，否则工程师会认为这些目标以及后面的功能需求都是你拍拍脑袋定的。他们不喜欢这种随意的需求，就像他们不喜欢随意定下的交付日期一样。你要慎重对待这件事情。

如果说目标是告诉别人你要做什么，那么非目标则是告诉别人你不要做什么。所以设定非目标非常有用，那些持不同意见的人能通过非目标来理解你为什么会这样规划产品。例如，你的设计团队如果担心你定义的产品必须拥有键盘才能正常使用，你可以告诉他们："移动端和无键盘支持是非目标。"

2.6.3　用例或用户场景

有些人把用例和用户场景分开单列。用例是指用简要的语句来描述那些用户必须执行的操作，用户场景则是指用叙述故事的方式来描述用户是如何体验产品的。

下面是一个 Google+ Hangouts 的用例：

用户能共享屏幕

再看一个更有意思的用例，它包含了上述用例：

当用户试图共享屏幕时，如果其他用户正在共享，该用户会收到系统让他确认是否替代目前正在共享的其他屏幕的提示。

用例描述非常精细，你不加思考便能读懂，工程师也能根据用例准确地判断该做哪些工作。因此用例（或者用户故事）在敏捷开发中发挥着巨大作用，每个核心任务都会描述成一个类似下面结构的用例：

作为视频聊天参与者之一，我希望能【共享我的屏幕给其他视频聊天参与者】。

这个敏捷模型强调的是用户类型与用户行为，在实际工作中非常有用。当用户行为趋向复杂时则更适合使用用户场景。例如，如果将 Hangouts 的用例重写成用户场景，你会发现开发者对预期的用户体验会有一个更好的理解。

乔迪希望共享她的屏幕。她点击了“共享屏幕”按钮。系统弹出提示要她选择想要共享的窗口或者共享整个桌面。每一个选项都有窗口预览图和标签描述，并且预览图是实时的，就像一个个小视频。当乔迪点击了其中一个选项后，她的屏幕就成功展现在视频群聊中了。但如果有其他人正在群聊中展示屏幕，系统会弹出提示：“瑞克正在共享他的屏幕，你想替代成你的屏幕吗？”如果乔迪点“不”，她则回到初始状态；如果乔迪点“是”，瑞克的视频界面将会切回到他的摄像头，乔迪的屏幕将会显示在群聊中。

无论是只写用例还是只写用户场景甚至两者都写，你都需要敲定它们的优先级。这样工程团队才能给工程任务定优先级并优化设计。说到优先级，我曾听亚马逊一个高管这样评论道：“优先级就是个扯淡的事！”离这种高管远点吧！如果不定优先级，资源有限的工程团队如何能够裁减功能以赶上发布日期呢？所以优先级非常重要！谷歌和亚马逊用的是相同的优先级规则，共分 4 级：

P0

没有该功能产品无法演示。

P1

没有该功能产品无法交付。

P2

锦上添花的功能。

P3

哈哈哈!

P3 的功能基本上要被裁减掉，甚至 P2 的功能也在裁减候选名单中。由于这套优先级规则也适用于之后的 Bug 处理，你的团队将开发出一套通用的词汇表和标准。衡量功能或 Bug 的重要性与紧迫度通常很难，因此建立通用的分级标准并尽早对它们进行分级是非常有益的。参考第 5 章了解更多关于优先级在交付后续阶段的应用。

有时候你认为一些用例无须在版本 1（V1）中实现，但团队成员又认为它们比较重要时，可以给这些用例加上 V2 的前缀或者其他标签，让他们明白这些用例会在 V2 中安排处理。其实 P3 差不多就是 V2，但列成 V2 的好处是让所有人都知道这个用例在 V1 中是不可能做的。但不管用例是放在 V2 还是放在 V1 中，你都需要现在就把它们描述清楚，这有助于工程和设计团队构思一个具备良好扩展性的系统，同时也有助于你少回答一些“呃，不过要是出现这种情况……”的蠢问题。

2.6.4 原型图或线框图

由于正在遵循一个行之有效的产品定义过程，因此你已经绘制了一些粗糙的原型图或线框图，将这些图粘贴到功能说明中，它们是用户场景的重要补充。

2.6.5 API

如果你还没写 API 文档，那就现在写，不过前提是已征得工程团队的同意。

2.6.6 负载规划

负载规划是指对未来一段时间内用户的使用量进行粗略估计并制订应对计划，这对工程团队来说非常重要。他们会根据你预估的使用量来确定哪些地方需要添加缓存，哪种类型的服务器和存储需要准备，哪些授权问题可能产生，等等。

预估使用量是件极其困难的事情。我曾在一个发布会上听到一个 Xbox Live 的负责人讨论他们的负载规划。他说他当时选了一个他觉得能在首年达到的最大值，结果他们最后做到了这个值的两倍。这在产品上是一个巨大的成功，但对他们的负载能力却是一个严峻的考验。

Xbox 预估的负载量虽然过高，但这并不是说它的方法彻底错了。在亚马逊和谷歌都是依据类似的方法来制订负载规划的。首先你会建一个表格来制订年度或季度负载能力规划。然后你需要预估存储量（帖子数量、图片数量、图片大小等）和流量（访客数量、访客停止时间、人均页面流量数）。在谷歌你还需要预估出口流量（从数据中心获取的数据量）和进口流量（你的服务器请求量）。包括亚马逊网站在内的大部分应用的出口流量都远远大于进口流量。但如果你构建了一个用户可以上传图片、视频或者其他内容的应用，你的进口流量就会非常大，所以请根据实际情况制订负载规划。

你需要维持适当的余量并预测一个余量值，这个值可能并不准确。例如，你可以这样说：“我设想需要 100% 的余量来应对超出预期的增长。”

你需要预估日均峰值，通常预估为均值的 3 到 4 倍是安全的，因为这个值代表了美国不同时区用户的重叠峰值。如果你的产品与众不同，比如是一个设计成开机启动的软件升级系统，你的峰值则会远远大于均值，所以请根据实际情况调整预估数值。

如果业务面向全球，还需要考虑如何缓解延迟问题，是部署多个数据中心，还是使用阿卡迈（Akamai）等公司提供的内容分发网络（CDN，Content Delivery Network，也称为边缘缓存）。

你需要制定预案以应对突发的高峰，比如在产品发布时期剧增的访问量。这时期的用户行为极为不同，你需要采用不同的应对计划。比如当《60 分钟》[①] 报道了你的产品时，你该如何应对突增的用户量呢？我在一家创业公司时曾经历了这样的事情，不过好在尽管延迟增加、网页做了分页处理，但产品始终是可以访问的。虽然并不完美，但至少我们扛住了。

你需要制定备用策略以应对最坏的情况。一次分布式拒绝服务（DDOS，Distributed Denial-Of-Service）攻击、一篇《华尔街日报》的报道或者一次数据中心故障都可能置你的产品于最坏的境地。不过灾难并不可怕，可怕的是你没有任何准备。你应该适当部署一些有效的危机管理系统，如流量限制系统、“系统目前繁忙，请稍后再试”的出错页面或者保存在 CDN 上的应用静态版本。

你可以在撰写负载规划这节时与工程团队就系统设计进行充分的讨论。你需要了解当系统过载时是部分彻底不可用还是整体被拖慢？系统是支持纵向扩展（指你每增

①《60 分钟》是美国哥伦比亚广播公司（CBS）主打的一档电视新闻杂志栏目。——译者注

加一台服务器，你就能获得这台服务器的负载能力）还是非线性扩展？

总之，关于负载规划你要做的就是进行合理的预估，你的工程主管则负责根据预估值来构建一个能稳定运行的系统。不要花太长时间在预估上面，只需花几个小时写个初稿，再找几个团队成员讨论下并将讨论出来的数值翻倍就大功告成了。

2.6.7　依赖

你需要将全部依赖方及其负责人列出来，如果有应急方案也一并列出来。功能规格定稿后你也应该发给各依赖方的负责人，让他们知道你需要他们的支持。他们可能只会阅读你的简介部分，不过这就够了，因为这部分已经清楚阐释了你要做些什么。

依赖不需要描述得太详细，你只需简单说明我们需要这个依赖的原因以及依赖会受到什么影响，比如流量或者异常情况。例如当我负责 Google Pack 时，我列出了以下依赖：

下载服务（dl-eng@ 负责，m_@ 接口）：我们需要下载服务托管已签名的第三方二进制文件，我们会每两周推送一次更新，然后已签名客户端会通过 HTTP 请求有效负载，所以不需要大量的 SSL 流量。我们只通过 SSL 请求 manifest 文件，manifest 文件会包含二进制文件的签名信息。我们会尽量避免临时的紧急的更新，但一年还是可能会有 1 至 3 次。

2.6.8　FAQ 和开放问题

你可以直接将 FAQ 和开放问题的链接地址放入功能文档中，也可以把内容复制过来，不过我建议最好保持 FAQ 的独立性，这样就不需要维护多个版本的 FAQ 了。

2.6.9　关键事件

你可能有一些硬性时间要求（比如苹果有它的世界开发者大会，又比如你的资金只能支撑到某个时间），这些时间都需要放入文档。你最好能列出主要事件的达成时间，如特性完成时间、可信测试者版发布时间，如果具体的工程量尚未评估出来，那预计的时间应该保守一些。在这一部分应该把重点放在硬性时间而非工程事件，另外再放上项目计划（参考第 5 章）的链接。

2.7 第 7 步：找出边界情况并得到团队认可

最困难的部分已经过去了。虽然你写的是一份没有人能全部读完的超级大文档，但每块内容都会有一些利益相关者关注，并且在写的过程中你也清晰地理解了产品的愿景。同时这份文档也是一份出色的职业材料，它会让你在未来跳槽时更受潜在雇主的青睐。目前一切都做得很出色！

接下来你需要和团队一起细究文档以找到所有的问题。在这个阶段你可能不得不承受尖刻的批评并做好变更的准备。不要为此焦躁或沮丧，你应该趁此机会好好与你的工程、设计、业务团队磨合。没有人能在第一次就创造一个完美的产品，否则还需要团队干什么呢？所以深呼吸迎接挑战吧！

你的团队将开始寻找*边界情况*[①]或者*极端情况*[②]，即极少出现的产品行为或场景。不要抱怨这个看似繁琐的事情，如果不找出所有边界和极端情况，你就无法采取应对措施，它们就迟早会给你带来伤害，现实世界中尤为如此。

Motricity 的资深项目经理、曾为摩托罗拉等客户管理过大型项目的艾伦·阿布拉姆斯认为，发现边界情况的最好方法是“漫步于功能之中”。确实如此，你需要时间来仔细地、创造性地思考用户会如何弄坏你的软件或者在某种意义上没有按照你的预期来使用软件。当你“漫步”时，请将想到的所有可能的边界情况以及应对策略写在 FAQ 或者产品需求文档中。

除了确保处理好边界情况，你还需要确保工程和用户体验团队认可你的产品规划。最好的做法是在开始时就把产品需求文档发给开发主管、测试主管以及用户体验主管。如果有其他主管也对产品感兴趣，如客户支持、法务、公关等，也一并发给他们。

文档发出去后，他们未必都有时间阅读你的文档（通常不止 1 个人没有时间）。你最好邀请他们开一个类似于设计评审的会议（第 10 章有更多关于设计评审会议的说明），给他们充分评论这份文档的机会。这些主管经常会贡献一些富有启发的独到见解，还能进一步指出一些你需要应对的边界情况。

第 7 步充满风险。一方面你必须承认并整合所有你的团队找到的边界情况，另一方面还必须捍卫产品的核心原则。如果产品有硬伤，你将很难得到工程团队的认

①edge cases，指在一个最大或最小运行参数下发生的情况。——译者注
②corner cases，指在多项运行参数同时处于极端水平下发生的情况。——译者注

可。而工程团队都不买账的产品，还指望你的客户会买账？工程师虽然看起来像一群穿着睡衣、无视专利的怪人，但并不意味着他们不能是精明的客户。所以你必须认真倾听并考虑他们的意见，同时尽你所能去说服他们，让他们相信这是一个令人惊叹的产品！

如果团队认可你的观点则万事大吉，如果他们仍不认可，那么解决方案也很简单：重复第 7 步，从使命开始一步步想清楚问题到底出在哪里，然后调整产品规划直到他们认可为止。你并不需要团队的每一个人都相信你的规划是完美的，而是需要他们同意朝一个方向前进并把产品视作是一个极有可能成功的实验！一旦得到了团队这种程度的认可，你就可以进入第 8 步了。

2.8 第 8 步：客户测试

拿客户做测试听起来不像个好主意，但亚马逊和谷歌实际上都在这么做。他们会向部分客户发布一些实验性功能，然后监测他们的使用情况。除了客户测试，亚马逊还会专门雇佣一些测试团队，而谷歌则坚定不移地奉行单元测试（不是功能测试）。但尽管有这样那样的测试，产品出炉后依然会有大量 Bug 存在。不过我在这里主张的“客户测试”并不是让你马上把软件开发出来然后扔给客户挑刺，我主张的是去找一批现存的或潜在的客户，向他们介绍你的产品设想和原型，并听听他们的反馈。

这个测试可以避免你做出一个没人想用的产品或者遗漏一些核心功能。谷歌高级副总裁艾伦·尤斯塔斯曾指出，团队会轻易陷入一场为莫须有的客户问题构建完美解决方案的狂欢中。所以你需要这个客户测试来验证你的目标、非目标以及优先级是否合理。

有些人把这种测试叫做“焦点小组”，也有些人把它叫做“试售”或对现存客户的“路线图演示”。你的用户体验团队或许会认为这个过程是一次认知性遍历：客户通过你对产品原型图的介绍来体验产品，然后提供关于功能和实用性的反馈（后面有详述）。以上林林总总的叫法或做法并不重要，重要的是你去做了这个测试！所以在产品演示文稿准备妥当之后你应该马上安排持续 3 周、每周 3 至 5 次的面向潜在客户的产品演示。

如果你手头没有现成的客户，那就让用户体验主管去做些基础性的用户研究：他

会邀请一些潜在客户来体验你的产品，并通过面谈来了解你的产品是否适合他们。有时候用户体验研究者会在美国最热门的分类信息网站 Craigslist 上招募一些志愿者来参加研究，并提供 100 美元的亚马逊礼品券作为回报。如果与你共事的人中有市场营销人员，你也可以让他帮你邀请一批客户来参加测试。如果实在找不到客户，那么家人和朋友也可以，但千万不要让参加测试的人大半都是极客。

举一个经典例子来说明为什么客户测试对你的产品方案如此重要。我在亚马逊曾负责一个叫做“实名制”(Real Name ™)[①] 的项目。我们希望通过引入责任机制来遏制用户在评价中肆意吹捧或者抹黑商品。简单来说，就是在用户提交评价时我们要求他提供真实姓名，并要求他输入信用卡信息以验证名字真伪。目前在 Amazon.com 上还可以体验到这个功能。

在公司内部我们就方案的一些细节进行了激烈的争论，比如是否应该允许用户继续以假名发布商品评价。由于久久相持不下，我将这个问题抛给那些最频繁写评价的客户，当然会先让他们签署一份保密协议。那些客户在收到我们的问题后，表现出极其强烈的反对态度，甚至有一个客户直接给杰夫·贝索斯本人写了一封措辞激烈的邮件。杰夫非常关注客户的意见，于是这封邮件迅速让我们团队达成一致，允许用户在评价中使用真名或假名或两者都用。我事后意识到如果没有这个客户的反馈，我们一定会经历一次悲剧的发布。当然我不是说有了这个客户的反馈发布就异常顺利，事实上也没有特别顺利，但至少避免了因为执意要求每个人填写真名并进行信用卡验证而可能引发的更大危机。

2.9　第 9 步：想清楚基本的商业要素——命名、定价和收益

到现在为止你的重心一直放在“客户是谁”“客户有什么问题”“我们该如何解决”上。这是真正正确的做事方法，只要你解决的是大众所共有的大问题，那么接下来的任务不过是小菜一碟。但如果你选择的是一个极小的利基市场，那么下面这些步骤会让你最终意识到你的错误。

①2004 年纽约时报的社论“评价的评价”对该产品有更多描述：http://www.nytimes.com/2004/08/03/opinion/the-review-of-reviews.html。

在这一步需要考虑的基本商业要素是产品命名以及产品能带来多大收益。当你向高管或投资者汇报产品方案时，需要一个确定的名称来确保你们讨论的是同一个东西。你还需要告诉他们产品能带来多大收益，从而使他们更认真地对待你的方案，而要想预估产品收益就得先给产品定价。因此现阶段你只需要考虑命名、定价和收益，像销售培训、营销方案、发布技巧等可以放在以后讨论，你的汇报对象当下也不关心这些东西。

先谈产品命名。你需要一个客户喜欢的、能注册商标并通过版权审查的名称，后者可以让律师把关。事实上讨论名称是一件极其主观且争议较大的事情，能比它还夸张的也只有定价了。所以我建议你挑一个能描述产品是什么的名称就好了！

这里举个谷歌的例子。Google Hangouts 是 Google Talk 团队的产品。当初为了给这个产品命名，产品和工程主管开了不下十次会议来讨论到底该叫 Google Voice 还是 Google Talk 又或者其他名字，比如 GVC。最后，谷歌负责社交产品的 SVP 维克·古多塔把它命名为 Hangouts——他非常喜欢这个名称。这个例子提供了另一种产品命名的方法：把命名权委托给其他人。其实名称并没有那么重要，它再出色也不能帮你做成这个产品或者毁掉这个产品，所以不要浪费太多时间纠结于此，赶紧定一个！

再谈产品定价。前面已经提到定价是一件比命名更糟糕、更痛苦的事情！它看起来很科学——毕竟里面包含了一堆数字——但最终大部分定价都是拍脑袋出来的。你和你的团队成员通常得花大量时间捯饬 Excel 模型中的各种定价公式，因为你可能发现即使半价客户也不会购买，或者高管认为使用产品应该免费然后通过广告赚钱，你于是不得不推翻原有公式并重新捯饬。我总是乐观地认为，如果你的产品好，你的客户基数大，你满足的需求真实有效，那产品的初始定价对长远成功有什么影响呢？所以这不值得你耗费太多时间。

但也不能凭空给定一个初始价格，你需要讲出个所以然来。我不建议你花时间阅读 300 多页的深度经济分析报告，它们大多时候都毫无用处。你只需理解产品定价的三个基本方法：按成本定价、按价值定价以及对比定价。

软件业通常不适合按成本定价，除非你提供增值技术支持或售卖软件许可协议（SLA，Software License Agreement）。不过即使这样也不适合按成本定价，因为成本定价的一个最大弊端就是很难真正统计成本，比如投资成本、一线支持成本、工程支持成本、长远法务支持成本、市场营销等。也许你能通过记账来得到精确的成本，但

那只是昨天的成本，你如何能知道明天的成本呢？

如果打算按价值定价，可以去调研客户，看看产品在什么价位他们最愿意购买，在什么价位即使需要他们也不会购买。拿到这个数据以后，不管去问 MBA 还是去问高中生，都能得到关于最佳定价的答案。不过这个方法只是看起来合理，实际上并不具备可操作性。首先，你的产品还不存在，客户怎么知道是不是真的需要它？如果不知道是不是真的需要，他又怎么知道什么定价最合适呢？其次，客户很少会如实回答关于定价相关的问题，他们甚至还经常出尔反尔。让我们再看下一种定价方法。

对比定价比其他两种方法要合理得多，但它有两个前提：①有一个合理的比较目标；②假定市场是弹性的，即产品会在价格下降时销量增加，价格上升时销量减少（很多产品都是这样的）。所以你得先在市场上找到比较目标。当你的产品比对手功能强大时，收费就高一些，反之亦然。如果市场上没有类似的产品，或者你的产品大体上算一个新产品时，对比定价就不起作用了。

再给你一些针对高科技产品的宽泛的行之有效的建议。

- ■分析竞争对手。如果能够对比定价，你就有了一个好的起点。
- ■调研客户愿意支付多少钱购买产品，虽然他们不一定会说实话。如果你是对比定价，这个数据可衡量定价是否可靠。如果你打算彻底启用一个新定价，这个数据也可参考。
- ■简化初始定价以降低用户理解成本。Google Apps 的价格分两个：50 美元 1 年或免费（仅限 10 个以内用户使用），而微软 Live365 的定价五花八门，让人眼花缭乱，莫非这正是他们的策略之一？
- ■把已经满足需求的定价再提高一些。等产品正式推出后再想涨价就难了。
- ■不要在定价上争吵不休。通常有些等级更高、脾气更坏的人会对定价有强烈的主张。我建议你立即做出让步，然后继续推进产品的交付，因为定价不是交付的全部——它只不过是其中一个小小的步骤而已。加油！

有了定价之后你就可以建立收益模型了。我见过很多团队主管和高级销售卡在这个环节。等我自己做了这个事情几年之后我才知道为什么他们会被卡住：他们害怕拍脑袋的事情，而收益模型中 50% 以上的内容都是拍脑袋出来的。剩下 50% 中有一半是从那些免费的高德纳研究执行概要中剪切出来的市场研究内容，另外一半是一些凭直觉想当然的东西。那些市场研究内容可以给你的模型贡献一些数字，不过它们只是

让你的预测从“拍脑袋”变成“科学地拍脑袋”。既然收益模型就是一个拍脑袋出来的东西，为什么我们仍然需要构建它呢?

第一，不管是 VC 还是业务主管，你需要让他们感知到你规划的是一个多么重要的业务。因此你需要一个模型来向他们展示未来 3 年的月度收益预测。

第二，构建收益模型能使产品设想具象化并证明产品机会有多大。收益模型包含了你的市场研究、你作为消费者的直觉以及一些数学运算，它们组合在一起可赋予你深刻的洞察力。你可能会惶恐地发现你设想的 10 亿美元收益实际上最好也不过百万——不过这不正是你需要的评估吗?

第三,一个简洁的模型支持你任意调整变量并反复进行预测。你可以借此了解定价、支持成本、市场费用等各种财务维度是如何影响盈亏的，这有利于你做出理性的决策。

所以你无需担心收益模型大部分是基于假设的，只需尽力保持它的简洁性。当你向其他人介绍这个模型时，如果他们质疑你的假设，那不妨根据他们的意见进行调整，毕竟你做这个模型的目的是争取资金支持以尽快开发出真正的产品，而不是预测未来。

下面将教你如何构建一个非常简洁的收益模型。你可以从 http://www.shippinggreatness.com 下载电子表格的副本。

1. 估算买家总体市场规模。①在我负责 Google Talk 时，通过估算企业在视频会议、语音会议以及长途 IP 电话上投入的费用，我发现这个市场规模极具吸引力。②许多消费性产品会向市场推出免费版，即你可以免费使用大部分功能，但如果需要更大的存储空间或更高级的功能或一些有用的道具，你得为此付费。这种模式的特点是优先考虑市场规模，后面再考虑付费转化。举个例子，假如 Facebook 的用户量为 8 亿多，那么它的市场规模也是 8 亿上下。③分析报告能帮助你进行估算。如果做的是一个新产品，你可以从公开免费的市场研究摘要或者 VC 提供的报告中摘取一些数字进行估算。

2. 预估市场规模的增速。它将是你的基线。当市场规模扩大时，你的销售规模也应扩大。

3. 估算你的目标市场占总体市场的比率：①估算你针对的细分市场（比如小型企业、中型市场或者大型企业）占总体市场的比率。②估算你可触及的国家的市场规模占总体的比率。你的产品可能一开始只在美国推出，但世界那么大，扩展到其他

国家将带给你巨大的价值。

4. 估算通过市场推广你能触碰到的用户规模。你可以把预估的市场预算除以关键词的千人印象成本（CPM，Cost Per Mile/impression）来简单估算出该值。

5. 预估触碰产品的人中会有多少转化成产品用户。

6. 找到其他新用户增长渠道并加入到模型中。假设你的产品中有一套“邀请朋友”机制，你便可以预估它的转化率并加到模型中。

7. 最后，将产品定价乘以每个时期增长的用户数便是收益了。如果产品是付费订阅类的，你还可以预估一个续费率然后计算利润。

在下面这个简单模型中（图 2-1），我假设我们正在销售一款名为“愤怒的山羊”的游戏（嘿嘿，我们是“快速模仿者”）。它不需要付费订阅，但它提供了在游戏内需付费购买的道具，这是我们收益的真正来源。在我的第一版模型中，我设想用户可以免费下载这款游戏，然后我们依赖应用内购买获利。

变量				
总体市场规模	600,000,000			
对该类游戏感兴趣的细分市场规模占比	20%	目标市场		
细分市场中能理解英文的规模占比	20%			
市场推广转化率 ($/用户)	US$1			
应用价格	US$0			
每用户支持成本	US$0.25			
应用内购买用户占比	20%			
应用内购买平均金额（付费用户生命周期内）	US$3			
病毒式传播转化率 (好友推荐)	15%			
收益预测	**Q112**	**Q212**	**Q312**	**Q412**
用户				
媒体引入的用户数 (每季度发布一次)	150,000	20,000	15,000	25,000
市场推广费用	US$100,000	US$100,000	US$100,000	US$100,000
AppStore常规带来的用户数	30,000	25,000	30,000	35,000
病毒式传播带来的用户数	0	42,000	28,050	25,958
总的新用户数	280,000	187,000	173,050	185,958
目标市场占比	1.17%	1.95%	2.67%	3.44%
收益				
应用售卖	US$0	US$0	US$0	US$0
应用内购买	US$168,000	US$112,200	US$103,830	US$111,575
总收益	US$168,000	US$112,200	US$103,830	US$111,575
成本				
市场推广	-US$100,000	-US$100,000	-US$100,000	-US$100,000
用户支持	-US$70,000	-US$46,750	-US$43,263	-US$46,489
总成本	-US$170,000	-US$146,750	-US$143,263	-US$146,489
利润/亏损				
季度	-US$2,000	-US$34,550	-US$39,433	-US$34,915
年度				-US$110,897

图 2-1 基本收益预测

我喜欢把可编辑的单元格标记为黄色（本书印刷版中会显示成灰色）以便快速识别模型中哪些是预估值而哪些是计算值。比如图 2-1 中的模型大部分数据都是预估值。这个模型告诉我即使占据 3.4% 的目标市场份额，“愤怒的山羊”这个游戏也没有多少赢利可言。你是否注意到我不是只预估了市场份额这一数值，我还预估了用户增长情况并将它同市场份额交叉验证。我认为这样做更加直观，它不仅告诉我们的用户会增长多少，还告诉我们这些增长从哪里来。

在汇报“愤怒的山羊”这个产品方案之前，我们还需要回过头来思考一下我们做出的这些假设。我构建的模型对这些假设（或者叫做变量）非常敏感。比如，互联网广告获取新用户的成本是每个用户 1 美元，但实际上我花费 1 美元获得的是 1.15 个用户（假设病毒传播率为 15%）。然后这些用户中 20% 的人会给我带来每人 3 美元的平均收入，或者说我能从每个用户那里平均获得 0.60 美元的收益。天啊！我的市场推广计划竟然导致每引入一个用户就净亏损 0.40 美元！

幸好改动这个模型很方便，而且现在改动总比之后被新来的在线市场营销总监缩减预算要好。为了赢利，我预计要减少那些成本高但转化率低的广告投入。换句话说，我需要降低目前广告竞价的最大出价以使我们的广告在转化成本可接受时才出现。我还需要修改产品定价模型，把应用从免费下载改为需要 0.99 美元购买，这样做虽然会导致用户转化率下降(我估计最多50%)，但会为我们赢得更大利润空间。图2-2展示了我们改动后的模型[①]。

现在就差不多了。我们正在缓慢增长。虽然没有加入运营成本，但单纯从产品视角来看我们是赢利的。接下来我们还需要进一步降低支持成本，目前约 20% 的收益都花在了这上面，对于一个游戏来说这太高了。更为重要的是我们还需要找到更多低成本扩大用户量的办法，比如病毒传播功能能否再优化一下？

你可以清晰地发现该模型对假设是高度敏感的，这不是它的问题而是它的优点，

---

①作者实际上改动的不止以上各点，还包括以下内容。

1. 图 2-2 中假定该版本不再只推向英文国家，而是面向全球，这使得目标市场的规模扩大 5 倍并且每季度的新功能发布会引来更大规模的用户。

2. 图 2-1 中应用内购买总体收益是“用户数应用内购买用户占比应用内购买平均金额”，图 2-2 中是“用户数应用内购买平均金额”，也就是说图 2-2 中应用内购买平均金额（2 美元）的概念等于图 2-1 中应用内购买用户占比 × 应用内购买平均金额（20% ×3 美元 =0.6 美元）的概念，之间之所以相差了 3 倍多是因为付费购买应用的用户再购买应用内道具的可能性更大。——译者注

因为它揭示了你的假设以及特定变量的重要性。一套合理的简洁的模型能帮你理清逻辑、明确核心指标、理解商业模式并最终赢得管理层的支持。但你仍需注意不要花过量时间在它上面——真实的用户和真实的数据永远比预测更有效。

我知道有些 MBA 学员在看到如此简陋的表格时会不屑一顾。它确实很简陋，你还可以添加大把的内容进去。但实践的真相是你只是一个软件团队主管，且在这个阶段你需要更多内容来支持你进行聪明的决策。你只需要“科学地拍脑袋”，而这份电子表格能帮你很好地做到这点。

变量				
总体市场规模	600,000,000			
对该类游戏感兴趣的细分市场规模占比	20%	目标市场		
细分市场中能理解英文的规模占比	20%			
市场推广转化率 ($/用户)	US$1			
应用价格	US$0			
每用户支持成本	US$0.25			
应用内购买用户占比	20%			
应用内购买平均金额（付费用户生命周期内）	US$3			
病毒式传播转化率 (好友推荐)	15%			
收益预测	**Q112**	**Q212**	**Q312**	**Q412**
用户				
媒体引入的用户数 (每季度发布一次)	150,000	20,000	15,000	25,000
市场推广费用	US$100,000	US$100,000	US$100,000	US$100,000
AppStore常规带来的用户数	30,000	25,000	30,000	35,000
病毒式传播带来的用户数	0	42,000	28,050	25,958
总的新用户数	280,000	187,000	173,050	185,958
目标市场占比	1.17%	1.95%	2.67%	3.44%
收益				
应用售卖	US$0	US$0	US$0	US$0
应用内购买	US$168,000	US$112,200	US$103,830	US$111,575
总收益	US$168,000	US$112,200	US$103,830	US$111,575
成本				
市场推广	-US$100,000	-US$100,000	-US$100,000	-US$100,000
用户支持	-US$70,000	-US$46,750	-US$43,263	-US$46,489
总成本	-US$170,000	-US$146,750	-US$143,263	-US$146,489
利润/亏损				
季度	-US$2,000	-US$34,550	-US$39,433	-US$34,915
年度				-US$110,897

图 2-2　赢利的基本收益预测

2.10　第 10 步：取得上层的认可

如果是自己创业，那么你或许可以休息一下了，因为你可以自己批准自己的方案。除非你拿了风投，那就没有这么自由了，一个投资了 1% 的人也会认为他有资格

知道你打算把他的钱用来干嘛，尽管那些钱他们赚得毫不费力。

如果按照计划把产品卖给了你的开发团队，你就应该知道文档中存在争议或者谬误的地方并做了处理。为了让最后的产品汇报更容易一些，我建议你先花点时间预售产品。在谷歌，最优秀的主管都知道怎么做这件事情，因为预售可以让管理层在公开回应你的产品方案之前先了解一些背景。在类似亚马逊和谷歌这种人人都极度繁忙的环境中，预售是一个非常有效的技巧。

预售是一个非常直接的爬向食物链顶端的过程，为了让负责决策的高管最终认可你的产品方案，你需要预先争取中间每一级老板的支持，然后让直接向该高管汇报的家伙预先顺畅地了解你的产品概念。如果预售得不好，你可能会提前出局。更糟糕的是由于他们没有理解你的产品理念，自然也就无法准确地传达给最终决策的高管，而高管通常都是一边收发邮件一边和人交谈，所以他也没有心思仔细去探究其中的谬误，于是你会悲剧地发现你还没来得及汇报，高管就对你的产品留下了一些不好的或偏差较大的印象。

还有一种预售方式是“路过式”预售。趁着负责决策的高管站在走廊或者倒咖啡的时候走到他身边和他聊一两分钟你想做的产品，这时候你追求的不是一个决策，而仅仅是让他知道有这么一个事，这样你之后的汇报至少会顺畅些，同时你对他可能有什么激烈反应也心里有个底。

等到直接和高管汇报产品方案时你又得体会另一番折磨。以向杰夫·贝索斯汇报为例，我在亚马逊工作的时候坊间已到处流传杰夫是一个非常敏锐的细节帝。当你想做一个几乎全新的产品时必须得到杰夫的同意。我只向杰夫汇报过几次，虽然我很快就忘记了当时是怎么汇报的，但有一点我印象深刻，就是我从未有机会按正常顺序演示过我的幻灯片。杰夫会让你快速翻到后面去直接讨论方案细节。我在谷歌共事过的一位高管也是如此，不过他会在讨论细节一段时间后又要求返回到前面去看一下产品背景，而杰夫从不这样。

这种不让你按常理出牌的情况会发生在谷歌、亚马逊、VC 以及所有的给聪明绝顶的亿万富翁做演示的时候。不过 VC 还好一些，埃里克·施密特会在你演示时一直处理邮件，你都注意不到他的思维其实已经跳到后面去了。

因此，在向这种位高权重的人汇报产品方案时，首先请确保你了解产品的所有信息。虽然你注定做不到这点，但你需要尽力做好一些。万一被问到一些忘记了的事情

或者还没来得及去了解的事情时（比如事情发生在昨天而那时你在忙着准备演示），不要试着糊弄过去，继续保持你英明神武的状态并回答他们：“这点我不是很清楚，我之后会去了解一下然后给您反馈。”记住，你面对的是一群高智商的亿万富翁，他们能轻易嗅到你的谎言，就像嗅到车里放的屁一样。试着按照你的方式去承认这个错误没有什么大不了的，这不过是向他们证明了作为一个菜鸟你无法理解他们有多么优秀。

其次，他们想了解什么你就说什么。不要坚持你的顺序，带他们去了解他们想了解的地方。只要他们对你的产品理解是正确的，那就随他们的兴致，想看哪块就看哪块。当投资者们想要知道你这个产品能赚多少钱时，不要和他们说“嘿，我还没讲到那里呢，让我们先了解下团队每个成员的背景吧”，直接转到定价那页！如果你认为必须做一些铺垫，可以试着说：“我们马上就会谈到这个了，不过我希望先谈另外一些重要的事情。”不过说真的，还是直接转到定价那页吧，不要以为你对重要性的判断要好过这些家伙。

最后，请阅读 10.4 节了解并实践“综述单页”法。那些亿万富翁都喜欢这个方法，因为它可以让你就最核心的部分进行互动交流且无需浪费时间在不重要的细节上。

2.11 产品已经准备就绪，去构建它吧

在这个点上你已经找到了一个覆盖面广、重要性高的用户需求，你还提出了一个独特的解决方案。你能通过产品单页或者 10 分钟的演示来介绍方案，也能凭借功能规格（包括 FAQ、API 和依赖清单）来讲述产品的所有细节。你还努力构建了一个简陋却令人振奋的收益模型，它正是你做这块业务的原动力。

不过在庆功之前你还需要去见一些人。你要把这次见面看做是一次与意中人的约会，扣好衬衫，打理好头发，准备好风情万种的表情，然后去见他们，恳请他们的帮助。构建产品可是个艰难的部分，不是吗？

我非常关注这一点。任何一个聪明一点的团队主管都能发明一个“策略”或者一个合理的产品方案，然后欢庆成功，因为这个环节衡量你成功与否的是你写下的文字和会议上许下的承诺。但你无法一个人就把产品构建出来，真正的难点已摆在面前：驱动你的团队在现实的重重困境中依然构建出可靠的软件。我应该告诉过你必须在正确的时间点完成开发吧？而且你还必须让你的团队喜欢你和你的产品吧？

第 3 章

赢在用户体验 3

SHIPPING GREATNESS

用户体验不仅是产品的外观样式，它还是产品的使用方式。交付卓越产品就是指交付卓越的用户体验。人们若是不知道如何使用产品、不喜欢产品的外观或者找不到登录入口，这个产品就谈不上卓越。因此，就算你雇佣或借调而来的用户体验设计师再能干，你也不能把事情全部抛给人家然后坐等上线，你需要和他们通力合作以构建完美的用户体验。不过你的工作不是去解决所有的用户体验问题，而是确保产品尽可能提供最好的用户体验，这意味着你要让设计团队发挥出他们的最佳水平。

为了让设计团队发挥出最佳水平，你需要先理解设计，再让设计团队理解你。你可以从了解设计师的各种不同角色出发来了解设计。等你了解了每个角色是做什么的后，第二个需要了解的东西便是如何评估设计，了解了它你才能和设计师进行有意义的互动。等你了解了该怎么和设计师对话后，第三个需要了解的东西便是如何和每种设计角色有效地沟通，其中包括了解如何评论视觉稿以及如何向设计师提供反馈。第四个也是最后一个了解设计的要点是学会使用线框图或原型图来辅助沟通，你可以通过 Photoshop 或其他画图程序来绘制这些图形。

3.1 了解各类设计角色：用户体验，用户界面，信息架构，视觉设计，用户体验研究……以及角色模型

不同头衔的设计师专注的领域不同，即便头衔相同，设计师们也倾向于专攻某个细分的领域。因此尽管设计师可以处理你的任何需求，但你还是有必要了解他 / 她更倾向于向哪个领域发展，然后相应调整你的期望。

用户体验（UX，User Experience）关注的是用户如何完成任务以及该如何优化向用户展现信息的方式。通常用户体验设计师会通过制作流程图或原型图来说明用户体验，其中原型图是用来描述用户界面某一部分外观的图形。有时候用户体验设计师会制作*可点击原型*，它由多张原型图组成，一些原型图上有可点击的对象，点击之后会切换到另一张原型图。可点击原型可以让你在有限的场景下模拟产品的使用过程，加深对产品的体会。

用户体验设计师对信息架构（IA，Information Architecture）尤为关注。不同于工程架构，信息架构研究的是信息该如何在用户界面中呈现，而不关心底层的数据结构。例如在一个确认订单信息的表单中，所有信息都是关联在订单号或者客户的电子邮箱地址上的，因此系统最关注的是订单号，而设计师最关注的则是用户必须完成的主要任务：提交订单。你通常可以提这样的问题来思考信息架构："页面中最重要的信息是什么？"上面这个例子中最重要的信息是购买的商品及其数量和价格，而非订单号。信息架构专注于理解用户怎样才能接收到信息，而不是系统怎么才能正常运作。

通常关于信息架构的问题没有绝对正确的答案，因此团队主管需要密切参与到设计过程中来。作为产品主管，你应该比设计师更了解用户，比如你要做一个棒球网站，你可能知道用户关心球队新闻甚过关心积分榜，这样你便可以与设计师共同确立一个新闻优先级高于积分榜的信息架构。

用户界面（UI，User Interface）是用户体验的旧称，它更关注单个页面或屏幕的设计，是用户体验的组成部分。

视觉设计（VisD，Visual Design) 是关于如何通过一种既赏心悦目、夺人眼球又清晰明了的方式来展示内容的一门学问。视觉设计师往往具有较强的平面设计、排版和美术功底。他们根据既定的信息架构，使用调色板之类的工具增强或削弱信息在用户界面上的醒目程度。好的视觉设计师会基于栅格系统排列按钮、文本及其他控件以增强产品体验的一致性。这种通过在设计界面中添加固定间距的虚线来辅助设计的栅格系统为设计师提供了一套有序控制留白和内容的框架，从而提升了用户寻找内容的便利性和不同场景下用户体验的一致性。

用户体验研究（UXR，User eXperience Research）是用户体验的一个特殊组成部分，它专注于研究用户是如何看待你的产品的。用户体验研究者们擅长通过研究来获得关于产品成败的统计显著性数据，这些数据会提供给工程团队，虽然它们有些概念化但意义重大。用户体验研究者了解如何挑选参与者，如何构建一个有序且不带偏见的研究，以及如何指导用户在研究中避免带有偏见的反馈。不止如此，一个出色的用户体验研究者还会出具报告，说明用户体验中哪些部分有效哪些部分无效，这样的指导极富价值。遗憾的是用户体验研究者的工作并不包含提供问题的解决方案，这个任务落在你和用户体验设计师身上，不过要是他们有想法的话你最好

听一听。

“但是等等，”你也许在想，“我怎样才能从一组 5 到 10 个用户体验研究参与者中拿到具备统计显著性的数据呢？”答案是你可以通过比较所有提过的问题来建立显著性。假设共有 5 位参与者参与 16 项任务，其中 15 项任务都相同，只有 1 项任务不同，当参与者完成所有任务后，你获得的不只是 1 组 5 个不同的数据点，而是 5×16 个数据点，这样子显著性就建立起来了。如果你对这个逻辑有些怀疑，那也不奇怪，因为在选择个体参与者的过程中，你和用户体验研究者可能将较高程度的偏见带入到研究中，所以对个体参与者的评估需要慎之又慎。例如在我们位于西雅图的研究中，所有参与者会因为咖啡和阴雨绵绵的天气而显得敏感、忧郁，于是我们采取了一些措施来调整参与者的状态以避免可能的偏见。

角色模型（Persona）是一个由雅各布·尼尔森倡导并备受欢迎的方法，这个方法提供给你和你的设计团队、工程团队一个评估设计的框架。你的设计和业务团队将创建一组虚拟角色来代表目标客户，这些角色模型拥有姓名、薪水和目标，你还可以赋予他们任何你知道的目标客户的特征，然后利用这些角色模型来评估设计的效果。例如在一个度假规划应用中，你可能会这样思考：“保罗是一个高级用户，他每个月都在使用这个工具，所以他不希望每次使用时都需要重新输入他的出发地址……也许我们可以帮他保存这个信息？然后还允许他修改？”

3.2 了解如何评估设计

许多软件从业者在刚开始理解用户体验设计时都会觉得无所适从，尤其是那些纯工程或商业背景的人。大多数团队主管从未接受过设计师的训练，也从未想过要成为设计师，但他们却莫名其妙地背上了用户体验的相关责任，他们被要求确保用户体验是“优美的”或“直观的”，最可怕的是被要求“要像苹果公司推出 iPhone 时那样尽善尽美”！是的，我就曾被这样要求过。

如果需要负责交付一套卓越的用户体验，你就必须问以下 6 个用户体验问题。你还需要理性、思虑周全地回答，以确保答案合乎情理。若能做到，你将最终收获一个设计精良的产品。记住在每次检查原型或设计时都要问这些问题。

6个用户体验问题

- 该用户界面要求用户完成的最重要的任务是什么？
- 这是最简单的解决方案吗？
- 信息是否组织得当？
- 设计是否易用且一目了然？
- 标准是否一致？
- 能否减少用户点击次数？

3.2.1　该用户界面要求用户完成的最重要的任务是什么？

面对一个新的用户界面时，你应该首先问自己：“主要角色必须完成的主要任务是什么？该用户界面要求主要角色完成的主要任务又是什么？”关注主要角色而非全体用户可以帮助你更好确定优先级。若以上两个问题答案一致，则设计是符合要求的，反之你就需要做些工作了。在某些用户界面下这两个问题很好回答，比如一些类似结账流程的界面，但对于棒球网站主页这类用户界面它们就不好回答了。你得通过充分讨论不同角色模型将如何体验这个用户界面，来找到这两个问题的答案。

在棒球网站主页这个例子中，你可以这样思考：高级用户保罗和临时用户查克都希望知道最新的比分，所以在信息架构中应让比分信息最为突出。新用户爱伦可能希望关注某个喜欢的球队，但我们知道爱伦是有更强定制欲望的用户，所以主要任务不是允许用户在主页上快速完成定制过程，而是让爱伦能在主页上快速发现有定制这样的功能。类似地，当高级用户保罗已经指定过喜欢的球队，我们必须给他提供一个快捷登录入口，让他可以登录查看他喜欢的球队信息，若已确认其身份，则应将预定制的用户界面呈现给他。

在这个例子中，把握好多个目标之间的平衡并将之清晰传达给你的设计团队至关重要。如果你想为那些骨灰级棒球玩家构建一个应用，你通过市场调研已了解到这些人是一群技术控，并且他们想要一些强大的工具，那么你可以告诉设计团队，他们需要关注的最重要的角色是高级用户保罗，然后是新用户爱伦，最后是临时用户查克，因为查克可以通过其他各类体育节目获取他关心的球队的信息。但如果你的网站是《纽约时报》，你的绝大多数用户可能便是查克这样的临时用户，并且他们还大部分还

来自纽约，因此你的角色模型按照优先级排列可能是：纽约的临时用户、其他临时用户、新用户、高级用户。

与设计师沟通时千万别这样说：“登录按钮太突出了，减弱些吧。”也不要这样说：“将‘你最喜欢哪支球队？’这个推广信息移到顶部去吧。”我们要做的是清晰地阐述我们的业务目标以及它们之间的优先级，之后将权力交给设计团队，让他们以此为基础进行一系列的优化。

还有一种可行的沟通方法是问一些直接的问题，如“为什么登录按钮在屏幕中间”。如果设计师回答：“我想让它非常明显！”你可以说：“你确实做到了！但我们的目标是优先满足来自纽约的临时用户的需求，根据我们设定的角色优先级，你做出的是正确的选择吗？”好的设计师会理解你的意思并相应调整用户界面。

设计、业务和工程团队必须紧密合作，共同定义每种角色的优先级。如果你无法清晰说明优先级，设计团队将缺乏工作热情，他们只能机械地按照你的意思设计，你最终得到的也会是一个有缺陷的用户体验。因此在面对一个新的设计时你应该先问自己以下三个问题：

- 谁是最重要的用户？
- 这类用户必须完成的最重要的任务是什么？
- 这个任务在用户界面中是最重要且最简洁的部分吗？

前两个问题是业务问题，它们为最后一个设计问题提供了背景信息。如果发现用户需要经历一系列复杂的步骤才能完成任务，或者需要折腾半天才能开启任务，你就必须停下来重新设计这块用户界面。如果这时候设计团队看起来没什么信心，那么很有可能是你要求他们兼顾太多相互竞争的优先级了，你需要返回去再次思考问题 1 和问题 2 的答案。

3.2.2 这是最简单的解决方案吗？

用户完成任务的能力与该任务的复杂程度呈非线性函数关系。换成简单易懂的话来说：你对用户要求得越多，用户完成的能力和意愿就越低，而且低得不是一点点。问问你自己，你解决用户问题的方案是最简单的吗？如果用户想把一篇关于某个棒球运动员的文章通过邮件转发给朋友，他一定要先注册账号吗？你不能先让他享受到转发文章这个福利然后再引导他去注册账号吗？后面这种方式用户更为满意，而且还能

省却不少步骤。当然它也会放大关于滥用的问题，因此你需要在这里做个聪明的产品决策。在这个例子中，过往经验告诉我们应首先着眼于可用性的优化，滥用的问题等它出现了再解决也不迟。我极少见到这种解决问题的方式失败过，而由于试着解决也许永远不会出现的滥用问题而导致产品开发步履维艰的情况我倒见到过。

前田约翰在他的《简单法则》一书中提出了一个简单化的框架。他把这个框架称做 SHE：简化（simplify）、隐藏（hide）和附加（embody）。与我之前的建议类似，前田主张“简化”特性，让用户只做他们必须做的，然后“隐藏”那些偶尔使用或者次重要的高级特性。其中一个隐藏这种复杂性的方法是把那些针对高级用户的特性放入一个叫做“高级选项”的对话框中，或者使用带收起展开箭头或“+/−”符号的选项框把它们收起来，不过要记住它们在界面中必须是可发现的。

如果一些特性可以用更简单的东西表达出来，你应将这个东西“附加”到特性中去，让它们并行展示。例如在 T 恤颜色选择器中，如果你只是提供一个文本下拉框，然后下拉框中每个选项都用文字来表达，用户就可能理解混乱。比如说当你看到“颜色：鲑鱼色”时，你会认为它是粉橙色（鲑鱼肉色）呢，还是银蓝色（鲑鱼外表色）呢？这恐怕得取决于你是否是一个素食主义者了。而解决这个问题很简单，你只需要给每个选项附加一张该颜色 T 恤的图片就可以了。

3.2.3 信息是否组织得当

有时候你想展示的信息会有多个行动点，你需要让它们保持平衡。亚马逊的产品详情页面便是一个经典案例。这张页面上的信息虽然数量庞大但组织优美，几乎所有内容块都统一按照它们的收益能力排序。有些特性的直接影响很难评估，如客户评价，它们被放到了页面底部。有些特性则很容易评估，如“看过此商品后顾客买的其他商品”，它被放在靠近页面顶部的地方。

在实际工作中你可能没有这样一个衡量指标来使你的设计变得简单（但工程师们却要头疼不已）。你需要深入思考你的数据和特性该如何合理组织起来。为了做到这一点，你应遵循以下条件。

- 最重要的客户类型最关注的信息应该最突出。
- 信息的排列方式应该像报纸文章那样从标题到摘要一一排列。
- 信息应该尽可能个性化且实时，也应在合理的前提下尽可能详细。当你能提供

“销量排名：1327”时为什么只提供“销量排名：前1000”呢？用户喜欢适度精确的信息。

- 最常用的控件出现在最容易找到的地方。

3.2.4　设计是否易用并且一目了然

当识别出了用户最需要完成的核心任务后，你需要问问自己这些任务是否是可发现且可理解的。可发现性是指用户发现行动点的能力。以“加入购物车”这个行动点为例，如果你的用户连“加入购物车”的按钮都很难找到，你这份工作也别想再干下去了。同样的道理，要是在你的负责下出现了用“+”号按钮来表示“加入购物车”这种事，你的设计不会通过可理解性测试，而你则会被扫地出门。

解决可发现性问题的方法有很多，以下为三种常用方法，你可以做些尝试。

1. 定位

在西方文化中信息的优先级是从左上角向右下角递减的。如果你想把行动点放在最显眼的地方，你很可能需要把它放在内容的左上角。

但这个原则也有一些关键的例外，其中一个便是“广告盲区”，由于用户总是发现网页顶部中央会放些“打猴子”[①]的广告，以至于会习惯性地忽视那个位置的内容。同样有很多网站会采用左导航的设计方案，如果你把基于上下文的行动点也放在左导航那个位置，也可能被用户忽视。

你可能曾听设计师说某某视觉元素是在“一屏以下”。这是一个老式印刷术语，指一些新闻报道位于报纸的下半部分，而由于摆在报摊上的报纸都是折起来的，所以这些新闻报道无法被一眼扫到。在网页浏览器中，折线位于浏览器与屏幕下方边缘交界的地方，在主流屏幕中浏览器从顶端到折线这一区域高约600像素[②]。iPad、Android以及其他一些设备由于屏幕分辨率不同而使得折线位置也不同。如果一项内容不在折线以上，那么它的可发现性就会急剧下降。

2. 视觉设计

视觉设计能有效解决可发现性问题，你可通过改变元素大小，使用差异化配色，

---

①类似于国内某些网站在页面顶部放的网页游戏类广告，因此国内用户也存在这个“广告盲区”。——译者注

②浏览器从顶端到折线这一区域其实就是网页打开后在屏幕中的可视区域，其他区域需要滚动鼠标向下拉才可看到。——译者注

或者跳出栅格来使你的行动点变得易于发现。不幸的是，视觉设计同样也会催生很多问题。我们知道让事物变美观的一个最好的办法是让它们简洁、平滑，比如说把汽车的门把手抹平。车子固然美观了，但车门也无法打开了。所以你需要特别注意那些华而不实的视觉效果，它们会削弱可用性和可发现性。

3. 惯例

应用程序、网站和企业都依赖于某种设计语言来使任务可被理解。例如在谷歌地图中，街道是用制图学中规定的白色和黄色来标注的。如果你的设计师把湖也标注成白色，这将产生混乱。同样，如果你随意对调对话框中“确认”按钮和“取消”按钮的位置，用户就会不断点错。因此，要想将软件做好，你需要让设计师先阐明惯例，并之后通过检查来确保他们始终遵循。

如果你对一个特性的可发现性和可用性存有疑问，一个最好的办法便是让真正的用户来测试。可用性测试可以告诉你用户能否发现你的行动点。另外，你可以通过利用 Google Analytics 这类工具监控目标点击情况来衡量转化率，还可以通过部署 A/B 对比实验来了解哪种设计的效果最好。

3.2.5 标准是否一致

惯例提供了某种设计上的简捷性，利用它，用户几乎能在你的用户界面中跳跃向前。例如 Mac 界面中“确认”按钮总是在用户界面的右下角，所以用户可以直接点击按钮而无须阅读按钮上方的文案或按钮的名称。但郁闷的是 PC 并不遵循该惯例，它的“确认”按钮位于右下角“取消”按钮的左侧。因此对于网页应用来说这个惯例就没有意义了。不过你最好确保你的应用程序中按钮始终位于同一位置，特别是当它们运行在 iOS 或者 Android 上时。

这里有一些惯例，你可以利用它们来使用户界面更易于理解。

- 所有主要按钮都应尺寸放大且配色一致。
- 一个用户界面中只有一个主要按钮。
- 使用一组按钮来表示“是”或“否”这样的选择。
- 不同优先级的行动点使用不同的样式。例如在 Amazon.com 上有“立即购买”按钮（主要行动点，我们唯一期待的）和很多较小的“了解更多”链接（次要行动点），其中按钮要比链接明显得多，而且全站都遵循这个惯例，包括“你的

账户”页面，这使得亚马逊这套体系运作良好。

- 当一个流程有 3 或 4 张页面时，告诉用户目前处于哪一步以及共有多少步。
- 在你的应用程序中使用下划线或者其他配色来强烈区分链接和普通文本。
- 遵守互联网 CSS 标准（如：鼠标划过链接时指针变为手的形状）。

3.2.6　能否减少用户点击次数

由于用户体验的大体流程已经确定，你可以着手去考虑如何减少用户点击的次数。比如你可能问自己：“我能把一个表单从两页合成一页吗？”用户必要的点击次数会极大影响用户完成这个任务的能力，亚马逊还持有一个与之相关的专利：“一键下单”购买（美国专利号：5960411）。

你还需要仔细考虑用户选项中的默认设置。如果你的默认设置符合用户的需求，用户就可以少点击几次，同时也少遇到一些异常结果。设计师通常将默认勾选的复选框称为“退出框”，默认未勾选的称为“选入框”。

另一个可减少点击次数的重要方面是减少用户在键盘和鼠标之间来回切换的次数。用户每次重新握上鼠标并找到屏幕上的指针都需要可观的成本。应尽可能减少这些切换事件。

3.3　了解如何与设计师沟通

设计师的工作很不容易，因为每个人对设计都有自己的观点。也正因这个原因，设计师很少会被当做专家来对待。然而要是把他们当做专家来对待，并专注于问一些得体的问题，你就能驱动他们交付一份极高质量的设计并帮助他们建立工作的主人翁意识。不过每个人都有不同的工作方式，设计师也一样。有些设计师会比旁人更敏感一些，有些设计师则要宽容许多。有些设计师也许乐于听到“感觉有点挤”这类评价，有些设计师要是听到你这样说会恨不得重击你一拳。所以请听取下方的沟通建议并做些变通，以迎合每一位独一无二的设计师。

1. 以用户的口吻说话

反馈意见时使用“作为【某种用户类型】我想……”这样的开头非常有效，Scrum 项目管理方法就是使用这套句式来创建“用户故事”以指导开发者编码。

2. 以提问的方式建立共识

例如，你可能问："iOS 中后退按钮有什么惯例吗？这个惯例是一致的吗？"你的目的不是让他们详细地讲述这个体验，而是通过问这个问题来使双方建立对设计原理的共识，你的团队将根据这个共识来形成具体的设计。

3. 反复讲述业务目标，如果有些目标相互冲突，则反复讲述它们之间的相对优先级。

帮助设计师了解他必须解决的问题是什么。设计师们每天要做出上千个决策，他们根据自身丰富的经验来优化产品。你能够帮得上忙的便是确保他们能理解你的目标。为此，要将目标具象化。例如："不应该要求大部分用户去滚动页面。因此，我们应该把输入框放在折线上方，对吗？"

避免设置主观目标也能帮助你的团队。像"用户需要在应用中体验到家的感觉"或"它需要让人感觉更友好一些"这类目标会让我感到不安。你怎么知道你已经达成目标了呢？是在可用性测试的参与者表露出了这种感觉并还小憩了一会的时候吗？不要提这样的目标，你应该寻找导致这类设计问题的根本原因。例如："我们正在要求用户在没看到任何价值之前就得在首页支付 10 美元。我们得想出一个替代目前体验的方案来，让用户先了解我们会提供给他们什么价值，而不是先填写信用卡信息。"

4. 用数据说话

统计用户点击数、浏览屏幕的数量和页面加载时间可使交流更为具象。应积极开展可用性测试。同上条一样，不要说些主观的东西，比如以"这感觉……"或"我喜欢……"开头的句子。

5. 提供一些竞争对手或类似体验中运作良好的案例

与设计师分析竞争对手的体验将帮助你创建一套集大成的设计语言。你也可以让用户体验研究者去考察其他竞争产品，并分析行业内的最佳实践。

3.4 学习如何借助图画进行沟通

有很多方法可以制作原型。其中一个最简单有效的方法便是在白板上涂涂画画。当你发现设计师好像在听你讲天书般茫然盯着你时，请转向白板画出你在讲的东西。你还可以进一步整理下你画出的东西，然后用手机拍下来。这些手机拍下的白板图画

是非常强大的沟通工具，而且易于制作。我见过一些团队非常喜欢这些图画的风格，他们会把这些收集的照片制成视频动画。

正规的原型有很多种形式，其中简单的一种形式是灰度线框图，它着重于文案、布局而非视觉设计，可用于展示你应用的结构。视觉稿是一种更为精致的视觉原型，它不仅能帮助人们理解每个元素的视觉分量，还同时为你的团队提供了一份详细的页面规格，尤其是引入了红线标注后。红线标注版视觉稿是一个用红色引线标注每个元素尺寸和颜色都十分细致的原型。最后一种主要的原型形式是可点击原型，它是线框图的扩展，也是构建成本最高的原型。可点击原型作用巨大，你可以在可用性研究中提供给用户使用，然后观察用户实际上会如何体验产品。

当在文档或演示中需要借助原型来传达想法时，我会首先使用线框图，因为它们的形式最简单。如果在原型中添加了太多细节，如颜色、图片以及其他华丽的东西，你会发现一些负责评审原型的人会陷入到对这些细节的纠结中去。制作线框图时需关注以下基本原则。

- 只制作用户界面中相关部分的原型。
- 总是使用完整的、经过适当编辑的文本。
- 控制花在视觉设计上的时间。
- 使用灰度色，不要使用其他颜色。
- 预期你的线框图会发生很大改动。
- 当心视觉花招。

只制作用户界面中相关部分的原型。例如你可以先制作一张完整的页面原型，后续只制作用户界面中发生变动的部分，如一个弹出的对话框和一个邮件验证通过的信息片段。这样做既可以节省你的时间，又可以避免创建出来的多个页面之间细节不一致，还可以免除每次调整页面时所需要的重复劳动。

总是使用完整的、经过适当编辑的文本。文本或“文案”对解释界面的意图有着极为重要的作用。因此对于大段的文本你可以使用设计师常用的“这里有一段话”之类的文字填充，但对于任何表单、按钮、对话框或其他有意义的控件你必须使用准确的高质量文案。这类文案可以帮助你的团队正确理解用户界面中不同元素的作用。文

案还是煤矿坑中的金丝雀[①]：如果发现需要写一大段文字来解释某个特性该如何使用，你就应该重新设计这个特性，因为用户可不会读大段的说明。

控制花在视觉设计上的时间。视觉设计、品牌、命名等元素都是主观的，与用户能否完成任务的关系也不大。不像文案，这些花哨的元素不会帮助你理解用户体验，要是你把它们添加到原型中反而可能产生关于样式的争论，而这种争论与你想要解决的问题一点关系都没有。你应该使用标签明确的占位符框来替代这些视觉元素，然后继续下一步。

使用灰度色，不要使用其他颜色。一般来说色彩会加大线框图的复杂度，并催生与视觉设计和品牌相关的各种质疑。参见之前关于线框图的提示。

预期你的线框图会发生很大改动。线框图非常适合快速沟通想法并可以促进讨论。当就线框图取得一致后，你的设计团队将开始构建高保真原型，但此时的线框图与你最初设想的会有很大不同。因此在制作线框图时，你需要考虑如何搭建才能确保后续能快速修改。当别人把你的线框图改得面目全非的时候你无需担心，这只不过意味着你正有力地推动着对话向前发展。

当心视觉花招。设计与编码不同，它可以轻易地要些花招来争取人们的认可。例如，使边角变圆或增加透明度会使事物更加美观。法务要求必须展示的退出框和法律文案会被轻易抛之脑后。为了网页或界面丰满用户会被顺手加上各种个人信息。当心这些花招。构建原型时需要同时考虑新、老用户分别会看到什么内容，同时确保设计出来的效果能够真正实现。

如果幸好有一位设计师帮助你制作原型，也许就不需要知道这方面的知识了。若你还能为了拥有同理心而去了解一些线框图的基础知识的话，那真是再好不过了。但软件行业大部分的主管总会在某些时候需要亲自绘图，这时候，设计师们发明的一些不错的快捷制作简单原型的方法就可能对你有帮助了。其中有两个大部分设计师都在使用的简单方法。第一个是使用如只能在 Windows 中运行的 Visio 或只能在 OS X 系统中运行的 Omnigraffle 那样的流程图绘制程序来绘制线框图，第二个则是使用 Fireworks、PhotoShop 或者画图工具来绘制较小的、高保真的、基于现有用户界面的

①金丝雀对一氧化碳、甲烷这类气体敏感，煤矿坑若是发生灾变，金丝雀会比人提早中毒死亡，相当于“生物报警器”。——译者注

改动。两个方法都值得了解，我们会在后面一一介绍。

3.4.1 使用 Omnigraffle 制作简单的线框图

我使用 Omnigraffle 制作线框图。它是一个非常出色的程序，在使用它工作的过程中你就能学到很多设计知识，它的创造者在塑造软件出色的使用体验上下了很大功夫，有不少亮点。Visio 也能完成类似的操作，但它缺少 Omnigraffle 一些好用的功能。

如果打算为用户体验的线性走查制作一系列幻灯片，你可能需要用到图层。图层就像描图纸，你可以让其可见、不可见或按任意方式叠加。你可以利用图层来创造通用元素，如将一张空白浏览器的截图作为单独的图层移到堆叠的底部，这样该截图就会显示在所有其他图层中，你的线框图也就看起来像是显示在浏览器中。为了避免在制作其他图层时不小心编辑到这个图层，你可以将该图层“锁定”。图 3-1 中我创建了一个带标题的浏览器模板并将其锁定。

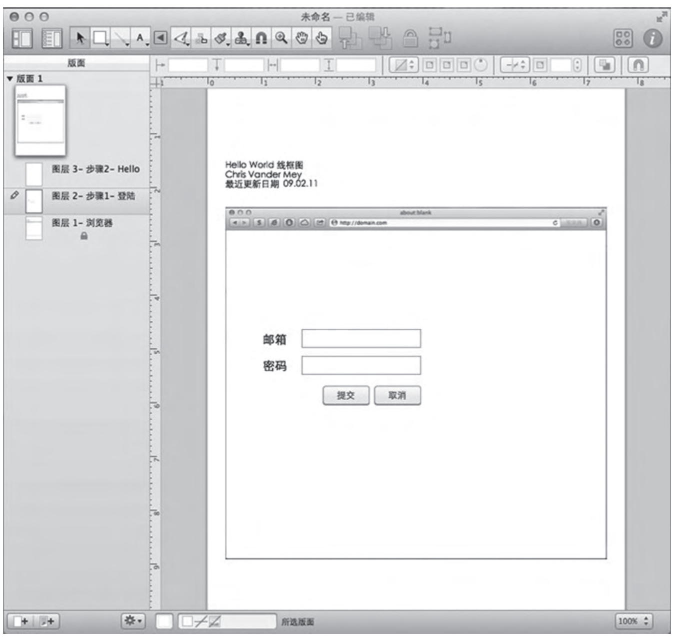


图 3-1 在 Omnigraffle 中创建图层

下一次，为你将要演示的每一张幻灯片创建一个图层。换句话说，为每次点击或者用户体验中每个步骤创建一个图层。

一个便于改动的基本线框图模板已经具备了，你需要扩展这个模板以继续增加它改动的便利性，比如可以增加一个通用头：创建一个新的图层，把它移到其他图层的最上面，这样它便能覆盖其他步骤（图 3-2）。然后将其锁定以便于后面的操作。

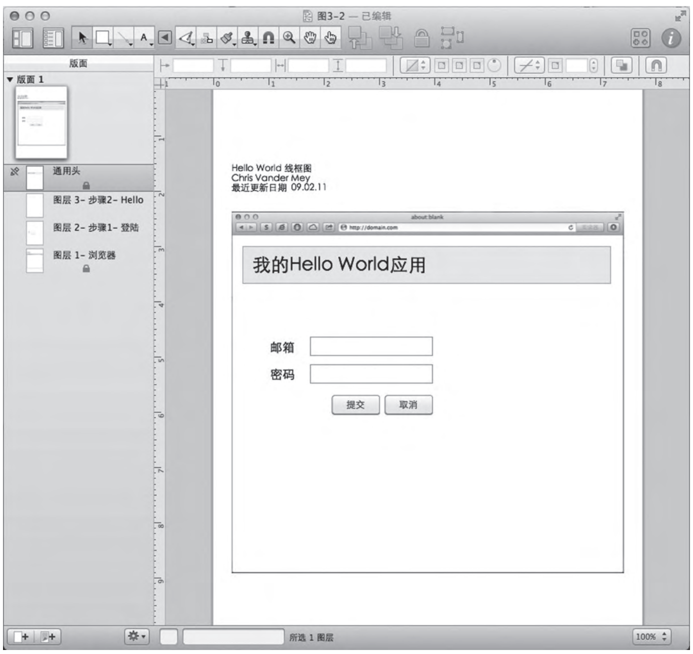


图 3-2　在 Omnigraffle 中制作通用头

现在你可以准备制作单个页面了。Omnigraffle 之所以在制作线框图上如此出色，有一个很重要的原因便是因为它的模板特性（Visio 也有类似特性）。模板是指一组可编辑模具（图 3-3），你可将用户界面元素拖入线框图中进行编辑。我使用了一个由 Konigi 出品的非常出色的线框图模板库，这个库中包含你可能会用到的任何元素。你可在这里下载它：http:// konigi.com/tools/omnigraffle-wireframe-stencils。



图 3-3 Omnigraffle 模板，陈列着各类按钮

将所有未锁定的图层隐藏，然后点击你想要编辑的步骤所在图层。将模板中的按钮、文本框、标签以及其他元素拖入到线框图中。

如果需要增加注释，你可以给需要注释的元素添加红色引线标注或者先前提到过的红线标注。该功能效果很好，对吗?

到现在为止我们关注的还是线性工作流，你也已经具备了制作演示幻灯片的能力了。但对于既定的任务，用户常常会有不同的处理方式，你需要说明这些异常情况。在这类例子中，你可以构建多个线框图并组合成流程图的样子。

要制作一个基于线框图的流程图，你需要像之前一样先绘制线框图，但不用考虑图层。图 3-4 是一个 Hello World 应用的简单示例。

然后添加第二步的线框图，用线把它与第一步连接起来，并各自标明是第几步（图 3-5）。你可能需要再增加一些说明文字来解释发生了什么。

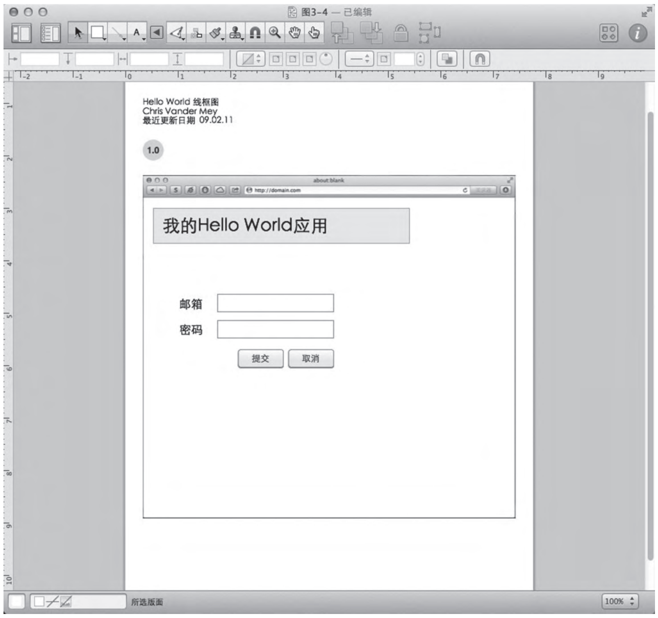


图 3-4　流程式线框图，步骤 1：绘制第一张线框图

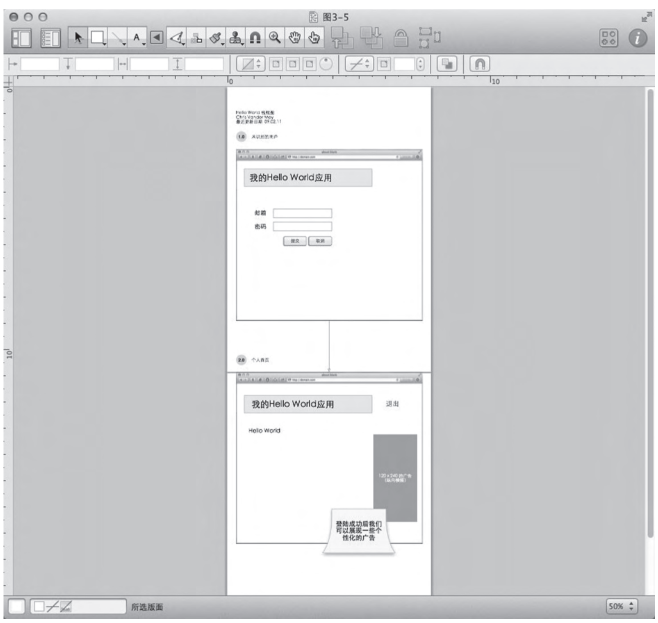


图 3-5　流程式线框图，步骤 2：连接线框图

这里有一个值得优化的地方：第三步你可以只绘制一个对话框，而不是绘制完整的用户界面（见图 3-6）。

Omnigraffle 是一个强大的工具，使用线框和简单文字元素构成的设计不但有趣而且易于改动。如果想构建更加复杂精致的原型，你可以创建可引用的组件作为对象，使改动应用起来更为迅捷。为对象设置属性和脚本可以创建用于测试的可点击原型。使用 Visio 或 Fireworks 可以实现许多相同的操作，并且 Fireworks 还能导出已构建成型的原型。不过你的目的是清晰快速地传达需求，这些高级的工具还是留给设计师们为好，你的原型应该保持简洁。

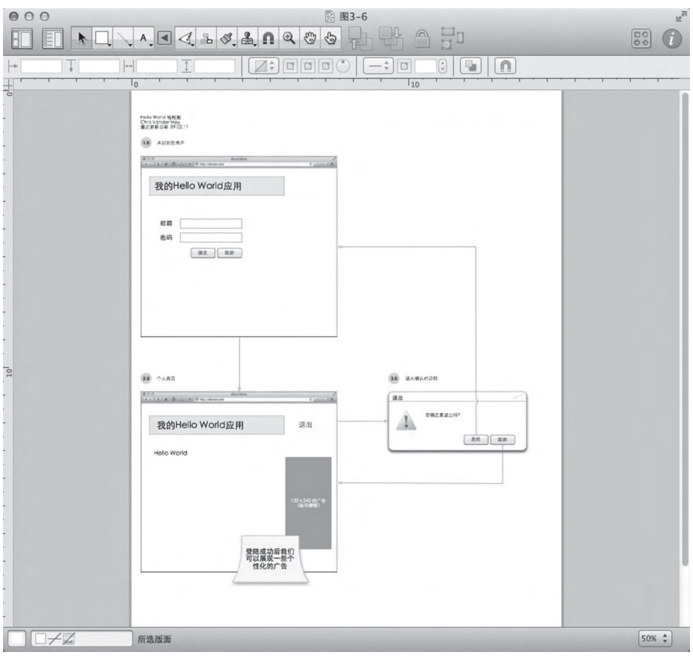


图 3-6 流程式线框图，步骤 3：绘制对话框，完成流程图

3.4.2 快速制作高保真原型

然而在某些时候线框图也没那么好用。或许产品改动很小以至于用线框图来表示

显得麻烦。或许你真的需要彻底弄明白某些特性的重要改动点。像这类改动我倾向于使用 Photoshop、Fireworks 或者其他图片编辑器，因为比起使用 Omnigraffle，我可以花费更少功夫做出一个高保真原型来。而且幸运的是，快速制作简单的高保真的原型并不需要你是一个 Photoshop 或 Fireworks 专家。

比方说我们想要将我们的 Hello World 程序放在亚马逊首页顶部 Kindle 的下方。首先你要将需要改动的用户界面截图截下来，用你的软件（在这些例子中使用的是 Fireworks）打开并调整画布大小以留出操作的空间。

使用矩形选框工具将需要改动或者需要腾出空间添加内容的部分删除。为了插入一个新的选项，我把需要改动的部分切出来并向下移（指针停留在矩形选框上，按住 Option 和 Shift 键并拖动鼠标[①]）。见图 3-7。

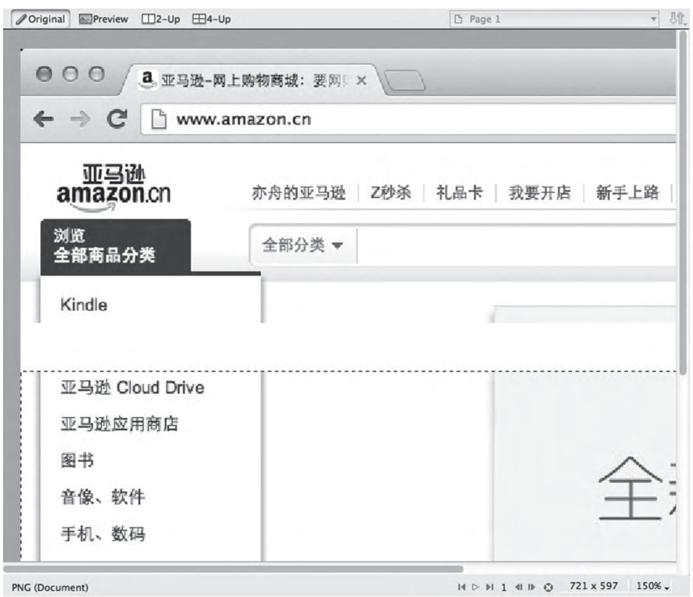


图 3-7 切开页面

若要复制基本元素，需要先用矩形选框选中它，然后按住 Command 和 Option 键

①OSX 中 Fireworks CS6 的快捷键为 Command+Shift+ 拖动。——译者注

并拖动到目标区域。在这个例子中我打算复制 Kindle 这个条目。复制好了之后，再把 Kindle 文字所在区域删除并用合适的背景填充该区域和其他区域（见图 3-8)。这里有一个小技巧，你可以继续使用矩形选框工具选中其他区域的背景色然后复制并粘贴到 Kindle 文字所在区域，而不是尝试使用油漆桶工具和橡皮擦。



图 3-8 复制和粘贴基本元素

添加元素。如图 3-9 所示，我添加了 Hello World 元素和一个绿色的“NEW！”提示符。你肯定不会做这么愚蠢的事情。(杰夫也绝不会允许。）我估摸着使用了一个相近的字体，但你也许知道你的产品使用的是什么字体。或者你还可以审查元素的 CSS！

清除下方空白并将图片缩放回正常尺寸。大功告成！（见图 3-10。）

这个例子本身微不足道，但你可以从中发现使用这个方法能让你在短期内制作可观数量的高质量原型。



图 3-9　添加你的具体内容



图 3-10　Fireworks 中制作完成的原型

第 4 章

赢在项目管理

4

SHIPPING GREATNESS

随着产品开发工作的进行，你需要总揽项目全局以便协调其他团队、安排发布计划以及确保各依赖满足要求。你需要肩负起这种程度的项目管理，同时还不能落下其他任何需要完成的事情。时间变成了稀缺资源，这让你的团队忧心忡忡，但你的薪水却有机会蹭蹭上涨。作为团队主管，你无法依靠 Microsoft Project 这类工具使一个复杂的项目模型变得易于构建和维持，也无法通过增加兼职人手夜以继日地工作来使项目在更短时间内完成。要想在交付上取得突破，你需要低成本的项目管理。

理解项目管理非常重要，所以我会在电话面试产品经理时问："你如何知道产品是否可以按时交付？"坦白说，这是一个伪问题，从来没有人能真正知道产品是否可以按时交付。但你可以预估它。因此，我的问题可以回答得很出彩，只要这个回答包含三项低成本的工作。

1. 创建一张简单的计划表并持续维护。
2. 跟踪 Bug，观察燃尽图，计算实现零 Bug 率（ZBB，Zero Bug Bounce）的日期。
3. 谨慎管理你的依赖。

无论是在瀑布式还是敏捷式的开发过程中，这三项成本低廉的工作都可以起到很好的作用。虽然你共事的每个团队都希望用不同的方法来管理项目，但这三项工作是必选的。快速掌握它们能让你在项目管理上无往不利。即便做不到这样，你也能通过它们了解哪些方面真的需要改变了！

4.1　创建一张简单的计划表并持续维护

你需要一张计划表来告诉你何时可以交付。一张简单的计划表只需包含任务列表和每个任务的工程评估量，这个量是指工程师或设计师完成该任务所需要的时间。你只需将这些任务按照他们认可的特性优先级排序并分配给团队成员，然后一张计划表就成型了。不要做其他画蛇添足的事情。

你会想当然地认为项目管理很复杂。于是你会因为不知如何管理敏捷开发过程中的待开发特性而烦躁不已，或者你会寄希望于 Microsoft Project 这样的软件来帮助你管理项目。我也是在被这些系统折磨了数年之后才认识到，对我和我的团队来说，一

张简单的 Google 电子表格就足以管理这些任务和评估量了。这张由我制作的电子表格包含了真正需要的所有东西。并且除了免费之外，你的整个团队还可以实时编辑它！你可以在 www.shippinggreatness.com 上找到一个工作示例，如图 4-1 所示。

接下来将介绍这份电子表格的使用方法。你可以在 www.shippinggreatness.com 下载，然后根据需要来决定要不要按照下面的步骤使用。第一步，你需要和开发主管合作将各项任务填入到任务分解区域。计划内的假期也应当做任务填进去（计算余量时不包含这块）。下一步，评估每个任务在不考虑余量的情况下所需的剩余开发者日，并猜测哪个工程师可以承担这个工作。将每个任务都归属到产品的某个目标版本中。你可能知道这些版本被称为“迭代”，其实它们也同样是你的发布版本。余量假设和你大致需要的开发测试比（例如，每 3 天的开发时间需要测试团队 1 天的测试时间）也必须填写。开发测试比取决于测试团队的规模。在图 4-1 展示的这个电子表格中，我加入了一些计算规则，以确保任务的截止时间点不在周末。

	A	B	C	D	E
7					
8	今日日期		2012-3-2	（使用 =TODAY() 来取到当前时间）	
9	假定余量		0.6		
10	编码时间/测试时间		3		
11	假定推送时间		周二或周四（2，4）		
12					
13					
14	发布日期				
15	版本	编码完成	测试完成	推送完成	
16	V1	2012-3-17	2012-3-22	2012-3-27	
17	V2	2012-3-26	2012-3-31	2012-4-5	
18	V3	2012-4-9	2012-4-14	2012-4-19	
19					
20	任务分配				
21	团队成员	V1	V2	V3	等等
22	chris@domain.com	6	4	0	
23	viki@domain.com	2	0	5	
24					
25	任务分解				
26	任务	剩余时间（日）	目标版本（或里程碑）	已分配工程师	依赖
27	克里斯的休假	5	V1	chris@domain.com	
28	邮件发送	1	V1	chris@domain.com	假期结束后
29	邮件格式化	2	V1	viki@domain.com	
30	定制邮件内容	4	V2	chris@domain.com	
31	反馈表单	5	V3	viki@domain.com	
32					

图 4-1　项目管理电子表格示例

因为这个模型在评估时使用的是“理想”的开发者日，所以在评估任务工程量时不必考虑余量，但在评估发布日期时就很有必要考虑它了。余量是一个“容差系数”，

用于容纳那些未曾预见的问题和一般的生产力损失所消耗的时间。一些团队判断五天的开发时间中只有三天是具备生产力的。剩余的两天则会发生各种各样的事情，最有可能的是一些联合会议、破碎的构建、配对问题以及失败的开始等。这些分心的事情根本无法预料，所以我发现 60% 的生产率估值非常合理。如果你的产品中有部分系统已经在运行了，你的效率甚至会更低，因为你要维护它们并服务现有的客户群体。早期项目则因为根本不存在什么 Bug 需要修复所以效率更高一些。还有一个关键的东西需要注意，这个 60% 的余量只假设了修复 Bug 的时间和不在计划内的私人事假，不包括计划内的假期。

我还在电子表格中增加了“假定推送时间”这个字段，用来建议工程师们不要在周五的时候把新软件推送到服务器上，这个建议很有必要。你肯定不想在午夜时分拼命去把那些在酒吧里买醉的工程师们抓出来去修复一个关于隐私侵犯的问题！另一方面，团队成员也需要固定的发布日期，这样运维那帮家伙才会友好地协助他们完成推送。在这个例子中，我假定团队只在周二和周四推送代码。

现在信息已全部填好，计划表也差不多成型了。下一步需要和开发主管来平衡每个人的工作量，并按照对发布日期的要求来调整各版本的任务数。查看电子表格的任务分配区域，你会找到那个剩余开发时间最长的工程师。这个工程师有时会被称为“长杆”或“处在关键路径上”。试着把他或她归属于 V1 的任务分配给其他不饱和的工程师。如果调整得当，你的工程团队的任务就会分配均匀，每个工程师的工作量就会相近。

现在你的计划更为均衡了，你可以把注意点转向发布日期了。如果 V1 看起来需要相当长的时间才能完成，而你又想尽早推出第一版或者想确保每两周就有一个版本发布，你可以把一些任务从 V1 移到 V2 去。优先移走 V1 中那些最不重要的任务，将任务分解区域中这些任务的目标版本值从 V1 改为 V2 即可完成移动，然后再把团队内的任务分配调整均匀。检查发布日期是否处于假期，以免造成严重的团队工作中断。当一切看起来都没问题的时候，计划表就完成了！

我喜欢这个表格，理由有很多。

- 它是我制作的。永远不要低估这种自豪感……
- 它便于你的团队更新剩余时间以及查看项目进展。谷歌文档中电子表格的可协作性还使得他们能在发现有遗漏的任务时立即添加进去。

- 它便于发现长杆工程师。
- 它便于配置或自定义。
- 它便于跟踪假期，因为假期在这里也是一个任务。
- 它便于挪动任务至后续版本。如果发现版本发布时间不满足要求，你可以将该版本中一些任务的“目标版本值”改成后续的版本号。你也能使用这个模型来跟踪里程碑。
- 它还适用于管理项目最后 30 天的冲刺。
- 它便于平衡团队内的任务分配。如果不想让克里斯在 V1 版本的关键路径上，可以将他的任务重新分配给维奇。
- 它便于预测项目各时间节点，包括编码完成时间、测试完成时间以及发布完成时间。现在你的测试团队知道什么时候可以开始新一轮的测试，你的市场部门也知道这款产品什么时候可以面世。
- 它预先将你的假设传达给了你的团队。
- “天”是一个很好的用于跟踪任务的度量单位。你也许常用“0.2 天”来描述那些非常小的任务，但我发现跟踪这些任务的最佳办法是放到 Bug 列表中。

虽然这张电子表格缺乏一些整洁的特性，如整合 Bug 跟踪系统或源码库，但这不是它的主要缺点，它唯一的主要缺点是无法跟踪依赖。我已经学会了如何处理这个问题，一方面是在每行任务旁边添加注释，另一方面是在每日例会上与团队进行讨论。

当项目所有主体工作尘埃落定，整个团队都集中精力于修复 Bug 时，我会停止使用项目计划表并转而使用 Bug 列表和 Bug 燃尽图。我们将很快谈到如何创建并使用燃尽图。

4.2 如何拿到评估量

一些经理发现找工程师评估工作量是件纠结的事情。而且有的工程师会过高评估工作量，有的会过低评估工作量。你要是没有和这个团队共事过一段时间，你将无法知道某个工程师会估高还是估低。为了能更轻松地拿到评估量并减少工程团队高昂的评估成本，你可以尝试下面的技巧。

1. 如果你不是工程经理，那么让你的工程经理去要评估量。

这点毋庸置疑。

2. 表面上接受评估结果

如果评估出来的工作量太大（超过一个星期），让工程师将这个任务分解成多个小任务。除此之外，向工程经理发发牢骚。

3. 认识到你的权力

如果你是工程经理，那么在评估这件事情上你是有绝对权力的。所以可以向团队承诺你会全力支持他们，而他们则要做出类似的承诺，即全力以赴支持项目。这很公平。所以放轻松点，告诉他们只要你还在这个团队你的承诺绝不会变。

4. 只跟踪剩余时间

我只跟踪一个任务还剩多少时间可以完成，其他如任务完成比例、任务评估用时与实际用时之比等我则毫不关心。评估用时与实际用时之比这个衡量指标，除了能识别出谁的评估最靠谱之外无法给你提供任何真正的见解，而你总要学习识别谁的评估最靠谱的方法。你还会发现缺乏经验的工程师几乎总是低估了所需时间，而富有经验的工程师则恰恰相反，因此你完全可以用这个经验法则来替代跟踪评估用时与实际用时之比。

只跟踪完成任务所需剩余时间是敏捷管理的一个原则，我很喜欢这个原则，因为它着重于项目的实际情况，便于你发现项目何时可以编码完成。

5. 要求不考虑余量的评估

你可以在项目计划表中加入余量这个参数并把它明示出来，这样你的团队可以知道你为可能（甚至必定）出现的问题提供了时间上的补偿。我看到关于评估是否该考虑余量有很多信仰层面的争论，但上述是我见过的最清晰的做法。

6. 每周一次在团队会议上评估各任务的剩余时间

每周评估一次可以防止你的团队被你频繁打扰，并让团队成员有机会了解并告诉大家为什么开发会快于预期或慢于预期。这个过程还有助于识别谁的进度受阻。

4.3　跟踪 Bug 并创建 Bug 燃尽图

Bug 燃尽图是一张反映你的 Bug 数量随时间变化情况的图表。它可以预测产品何时能够交付。制作燃尽图需要为不同严重等级的 Bug 各绘制一条其数量随时间变化的曲线。你还可能想要绘制一条描述 Bug 总量随时间变化的曲线。图 4-2 向你展示了一个示例的 Bug 燃尽图。

你应该期望接近编码完成时 Bug 数量会随时间不断增加，然后接近发布时 Bug 数量会随时间不断下降。这些 Bug 下降的比率，或者说这条曲线的斜率，被称作发现 / 修复率。当发现 / 修复率小于 1，即每天修复的 Bug 数量超过每天发现的 Bug 数量时，你才能确定 Bug 的具体规模并精准地预测发布日期。

当 Bug 发现 / 修复率降到 1 以下时，你便能通过计算 Bug 数归零的日期来预测产品何时能够按照给定的质量等级发布了。在图 4-2 中，你会发现按照这样的 Bug 走势，我们可以底气十足地告诉大家，我们可以在 10 月 31 日发布内部试用版本之前，修复完成所有最最严重的 Bug，然后在 11 月 15 日左右正式交付。如果你对测算出来的发布日期不满意，你只有两个选择：降低你的质量标准，或者增加工程人力以更快修复更多 Bug。

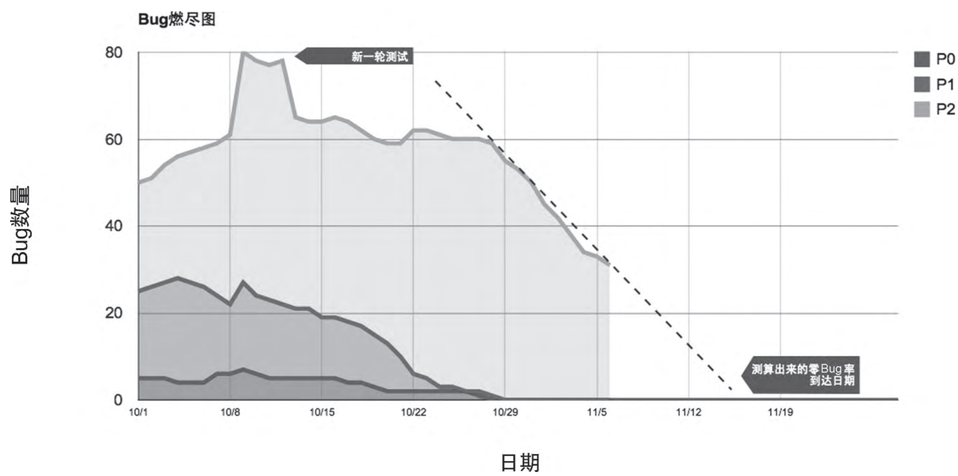


图 4-2　Bug 燃尽图示例

4.4　管理依赖

管理依赖并没有什么秘诀可言。你唯一能做的便是将风险最小化，而最小化这些由依赖引发的风险有一些关键技巧，我称它们为“五个如果”。

1. 如果去除它也可以运转，那就去除它。

少做少错是不言而喻的。减少特性总是能减少你的风险。在产品的早期版本中，将某些特性改由人工实现可以消除风险。例如将开发客户支持申请表单换成直接提供一个客户支持邮箱（形式为 mailto:xxx@xxx.com），诚然这会增加一些工作量，因为工作人员需要理清楚客户想要咨询的问题，但你并不知道到底会增加多少的量，所以你应该推迟对该特性资源投入以减免风险。

2. 如果内部能构建，那就内部构建。

在谷歌一些最高效的团队，尤其是 Android 和 Chrome，都强调要遵循“所有东西都构建在内部”的方法。事实上他们也是这样做的，虽然这会让其他人不爽，还会在某些方面拖慢开发速度，但它创造了一个允许高频率交付的环境。谁能抗拒交付的诱惑呢！

3. 如果必须添加一个依赖，那就趁早添加。

把风险最高的问题放在最前面去解决，这样才能更早发现风险，更早采取适当的矫正措施，从而增强按期发布项目的信心。

4. 如果必须添加一些依赖，那就依靠它上一个已构建的版本。

总有人会这样鼓动：“版本 2 的 Foo 会更加好用，它的开发团队也进展良好！计划用版本 2 吧，版本 1 实在是太丑陋了。”其实这样做基本上是不经济的。风险是交付的敌人。

5. 如果交付得早，被依赖伤害的可能性就小。

这条原则看起来有悖常理，实际上却很靠谱。你计划依赖的系统和产品总是在你眼皮底下悄悄地发生改变。例如一个你依赖的有利的业务关系会因为合作方雇佣了一个新的 CEO 而失效，而你根本无法预测这样的事件。因此，早点交付总能帮助你降低风险。

第 5 章 赢在测试 5

SHIPPING GREATNESS

如果你交付的软件无法正常工作，卖不出去是一方面，更糟糕的是你会因此蒙羞。这就是为什么我会对任何打算交付的产品进行“高中蒙羞测试”。这个测试很有效。在高中年代谁没有因为荷尔蒙引发的冲动而留下过深深的心理伤痕，好莱坞等电影行业还从中挖掘从而产生了巨大的社会影响力。你也可以利用这些心理伤痕，只需扪心自问：“我能确信当一个高中老同学看到我的产品时我不会感到羞愧吗？”这就是“高中蒙羞测试”的全部。

这个测试能帮助你确保你的团队是快乐的。汤姆·德马可和蒂姆·李斯特在他们合著的《人件：富有成效的项目和团队》中指出：“伤害团队的一个最佳方法是要求团队成员去做一些他们并不引以为豪的事情。”记住，你的工程团队成员都有一帮高中老同学，别让他们因为你的产品而蒙羞。

那么，如何确保你交付的软件不会让你蒙羞呢？你可以遵循下面 8 个主要步骤，这些步骤对产品质量有着重大影响。

1. 坚持测试驱动开发。
2. 围绕优秀的测试主管组建测试团队。
3. 亲自评审测试计划和测试用例。
4. 自动化测试。
5. 虔诚地推行内部试用（Dogfood）。
6. 开展找虫总动员。
7. 勤勉且有条理地处理 Bug。
8. 任命可信测试者以构建最后一道防线。

做好以上 8 步能帮助你在交付卓越软件的道路稳步前行。接下来让我们深入讨论细节。

5.1　坚持测试驱动开发

谷歌公司洗手间的墙上张贴着一句类似俗套小广告的话：“调试过劳死，测试嗨翻天。”这句咒语充满力量。调试需要你去解构、分拆软件直到找到问题为止，这简直

就是在通往交付的路上开倒车。测试驱动开发则可以帮助你确信你做的不是无用功，它还可以帮助你的团队在复杂的系统环境中生存下来，因为在你依赖的某个接口被其他工程师或团队损坏后，测试会立即报错。

你可以参考其他资料以全面了解测试驱动开发（参见附录 3），这里仅简要描述测试驱动开发的过程。

1. 埃迪工程师将代码分成多个片段，每个片段负责执行一些简单的操作。这些片段称为单元。例如，`countToTen()` 是一个软件单元。

2. 在写`countToTen` 这个方法之前，埃迪先写了一个测试，即单元测试。大体是这样写的：`If countToTen() is equal to 10, then pass;else,fail.`

3. 单元测试写完后，他开始写`countToTen` 方法，如果索引在循环中意外失效导致`countToTen` 实际上输出的是 9，测试就会失败。

4. 当软件构建时，所有的单元测试会自动执行。

挺简单，是吗？确实是，不过要想掌握它你还需要一些训练。测试驱动开发还有一个额外出色的点在于，发现回归 Bug 会变得更加容易，因为每个构建都会自动运行测试。你可以研究下 JUnit（面向 Java 的单元测试框架）这类软件以实现自动化构建和程序验证。

5.2 围绕优秀的测试主管组建测试团队

无论你的工程团队多么优秀、编写了多少单元测试，总是避免不了 Bug 的。找到这些 Bug 的最佳策略就是雇佣或者任命一位测试主管。测试主管是软件发布质量的主要负责人，同时也是产品经理、工程主管和市场主管的关键合作者。测试主管需要确保测试用例撰写准确、覆盖完整，且被正确执行。一个优秀的测试主管会不断培训缺少经验的测试人员并帮助设计优秀的测试自动化架构。如果你拥有一个真正强势的测试主管，测试团队的每个人都会减少妥协，追求完美，开发团队就不得不去创建更多更好的单元测试。

不止如此，你在开始阶段就需要一名优秀的测试主管还有另一个关键原因。测试团队和工程团队的团队文化通常是不一样的。测试团队无时无刻不在想办法找出问题。如果碰到一个典型的能力不足的工程团队，测试团队会每天抱怨不停且难以排解

负面情绪。测试团队和标准的工程团队在流程、规范和标准上也都不太相同。因此即便你把测试和工程团队融和得很好，也需要一个优秀的测试主管来帮助你管理测试团队，这对你有很大帮助。

测试主管还可以帮助你解决在雇佣测试人员时所面临的特有问题。越是优秀的测试工程师越难雇到，他们中的大多数人在技术上都有很深的造诣，因此他们更愿意开发自己的软件而不是去测试你的工程团队开发的软件。有两个办法可以让你跳出困境：降低标准雇佣测试人员并聘请管理者去管理他们，或者按高标准雇佣外包团队。两个方法各有利弊，我更偏好后一种。

5.2.1　选择一：降低标准雇佣测试人员并聘请管理者去管理他们

我不喜欢聘请管理者然后让他去雇佣一群平庸的测试工程师。我坚信这种方式会建立错误的动机。它强调层级，并且你那个聪明的测试经理还需要花费大量时间管理那些平庸的执行者们。当然，有机会做管理者对一些人还是有很强吸引力的。

降低标准雇佣测试人员这种方式还有更大的弊端，未来你是需要晋升这帮家伙中的一些人的。比如 5 年后，他们将认为自己有权晋升了，而你将面临一群二流的人才进入管理层的境况。常言道：“一流的人雇佣一流的人，二流的人雇佣三流的人”，很不幸它在软件行业是绝对的真理，你的生产力、产品品质将会因为雇佣了一群三流员工而直接下降。

5.2.2　选择二：按高标准雇佣外包团队

与外包测试人员一起工作会有些弊端。

- 你需要一个开发主管负责与测试人员的联系。这会增加工程团队的成本。
- 培训成本是沉没成本并且不可回收。
- 能力突飞猛进的测试人员期满离开后可以把在这边学到的知识用在其他地方。

但相比这些弊端，我认为它的优点更多。

- 在人员管理层面，外包的组织成本比全职雇佣低。
- 与外包商合作比与内部成员合作顾忌更少，你可以要求更快的速度、更高的产能以及稳定的质量。
- 你不需要晋升那些三流员工。

5.2.3 选择三：按高标准雇佣测试人员，不使用外包团队

这不算一个选择。谷歌曾经尝试过，但发现雇不到符合要求的人，于是要么交付后还有 Bug，要么工程师自己去做测试，项目到最后总是窘迫地收场。如果你还是要选择这个方式，那么后果自负。

5.3 亲自评审测试计划和测试用例

无论选择哪种方式组建测试团队，你都需要有人来撰写测试计划并且自己来确保测试用例的质量，这意味着你需要亲自评审并确认通过测试计划。

一个测试计划由很多测试用例构成，这些用例是从你的产品需求文档中派生出来的。如果你的产品需求文档写得十分糟糕，你的测试团队就无法测试。但如果你把自己的工作做好了，你的测试主管就能把工作做得一样好甚至更好。

测试计划通常是用电子表格创建的，因此你能方便地整理测试用例。检查测试用例是否包含下列描述性要素。

1. 领域

这一列描述哪部分的用户体验将被测试，你可以合并相近的项。

2. 严重性

该列定义了如果测试失败你会将此归为哪个级别的 Bug，通常有 1 ～ 4 级。

3. 前置条件

前置条件指定了测试人员在测试前必须做的事情。比如测试购物车中信用卡验证过程，前置条件应该是用户已登录、购物车已添加货品、送货地址已填写，然后测试才能开始。

4. 需执行的任务

任务由多个步骤组成，是测试的主要内容。任何步骤失败测试都失败。

5. 后置条件

后置条件描述了应用程序在任务执行完毕后所处的状态。以上一个例子为例，后置条件可能是用户看到一个包含确认编号的确认页面，同时信用卡会在 10 秒内扣除相应的金额。

图 5-1 展示了一个测试用例的示例。

	A	B	C	D	E
1	领域	严重性	前置条件	任务	后置条件
2	上传动物图片	S1	用户已登陆	1，进入首页 2，点击"上传动物图片" 3，选择附件"//src/giraffe.jpg" 4，提交	1，用户被重定向到动物详情页。 2，动物详情页展示上传的长颈鹿图片。

图 5-1 测试用例表格

如果时间不够宽裕，你可以每轮测试只执行高严重性的测试用例，这样虽然完整性有所欠缺但速度更快。这个方式也适用于验证一些微小的产品变更。你可以只测试发生微小变化的部分和高严重性的测试用例，这比全部测试一遍要省很多时间。在这里重新执行一遍高严重性的测试用例非常重要，即便你觉得这个微小的变更与其他特性毫不相干。你需要确保一些基本的特性不会意外停止服务。在单元测试覆盖率较低的复杂软件系统中，微小的变更很容易导致主要特性瘫痪，所以为了避免在老同学面前蒙羞，你需要防范于未然。

一轮全面测试后的输出物是 Bug 列表，有时候这个测试结果会让人诧异。这个时候很关键，作为团队主管，你需要一边向团队强调“坏的消息就是好的消息”，一边大力表扬测试团队的努力和成果，毕竟你还需要测试团队继续鼓起干劲寻找错误。如果你的团队在每轮测试后都士气低落，或者测试和开发之间的关系势同水火，产品潜藏的 Bug 就会越来越多，而找到的 Bug 却越来越少，这样的产品要是上线，你就等着被羞辱吧。

评审测试用例十分繁琐。你必须亲力亲为，即便只是为了维护与测试团队的感情。这里有一个小诀窍：固然坚持评审完所有测试用例是最理想的，且每一个注意到的人都会对你赞赏不已，但你也可以选择只关注以下三块内容。

1. 用户体验

确保测试用例覆盖了用户体验中最重要的部分，尤其是“新用户操作指引”流程和出错情况。

2. 安全和隐私

测试应该包括试图破坏你的网站。

3. 依赖

如果你依赖了一些非内部构建的数据库、第三方服务或软件，确保这些依赖将被严格测试。它们可能会在没有告知的情况下变更或损坏。

如果测试计划覆盖了这些主要领域，你便有了一个良好的开端。

5.4 自动化测试

还记得聘请一个优秀的测试主管是多么艰难吗？应对这项挑战的一个最靠谱的方法是找到一位愿意编写测试自动化程序的测试主管。如果你的测试主管能够精心搭建一套独立于产品代码的测试系统，你的测试工程师们将受益极大。更为重要的是，测试自动化程序会不间断运行，干着数十人才能干完的活。

你可能想：“哎呀，这得额外写多少程序啊！”不要担心，大多数测试自动化程序可以用脚本语言编写，它们也不需要多好的扩展性，而且还有现成构建好的框架可以使用，因此测试自动化的开发会更有效率。

作为团队主管你可能不需要亲自编写测试自动化程序，但你需要确保它正被不断地构建出来，因为你永远负担不起完成测试工作所需的全部人力，但你负担得起运行测试自动化程序所需的全部机器。

5.5 推行内部试用

微软主张“欲卖狗食，必先尝之”的理念，意思是说你应该把打算推向市场的软件先在公司内部试用。换句话说，不要自己团队吃人食，而让用户吃狗食。强制你和你的团队使用自身研发的软件，体会用户的痛，能帮助你们逐步提升紧迫感，理解用户困扰。亚马逊和谷歌都笃信内部试用的理念。

推行内部试用也会遇到挑战，特别是你要大家试用的软件已经有了一个比较好的、没什么 Bug 的替代品时。比如谷歌想让员工去试用谷歌文档，但大家都在使用微软 Office，这时候解决该问题的最佳办法就是停止在公司电脑上默认安装微软 Office，这不但能推动员工去试用谷歌文档，还能节省办公软件成本。

有时候降低试用的门槛也能起到推动作用。比如亚马逊为了让员工试用亚马逊金

牌服务（Amazon Prime）而专门提供了员工折扣。一些团队会提供 T 恤或 iPad 等奖品给报告不重复 Bug 最多的人。我曾经见到一个工程总监为了找到一个“非必现 Bug”（heisenbug）的必现方法而提供 5000 美元悬赏。非必现 Bug 是指一种不按逻辑出现和消失的 Bug，遵循海森堡测不准原则，这个 Bug 常常让人崩溃。总之，你需要想尽办法让内部试用成为你团队文化的一部分，即使别的团队没有这么做。

在内部试用过程中你还会发现一个有趣的现象，即你的 CEO 经常会遭遇一些别人从未遭遇过的糟糕体验。你第一次碰到这种情况的时候心里一定十分懊恼吧。你正想给这个人留下深刻印象，却事与愿违！不必苛责自己，你随后会发现其实每个团队主管都有过类似的经历。杰夫・贝索斯经常会发现一些稀奇古怪的 Bug，运行在拉里・佩奇电脑上的软件经常会出现故障。这样的例子还有很多，究其原因目前最被认可的一个是：高管们天生就是令人敬畏的试用者。

以拉里为例，你无法想象有多少早期测试版的软件安装在他的电脑上，有多少实验性质的程序运行在他的账号下，每个像你一样的团队主管都渴望让拉里确信你们的产品已取得重大进展，于是在拉里的电脑上会出现一些奇怪的状况也就不足为奇了。要想避免前面提到的尴尬情况，你需要搞清楚哪些东西会影响到你的产品并做好应对方案。

杰夫则不同，他会用一个全新的观点来审视你的产品，因为他之前从未见过这个产品，也不知道它能用来干什么。因此你最应该做的事情便是像他一样思考。装上异形大脑，取杯咖啡，清理浏览器缓存，格式化硬盘，试着忘掉任何关于产品的东西，然后开始使用。

如果你决心虔诚地推行内部试用，下面列举一些你可遵循的且会对你会有很大帮助的最佳实践。

1. 计划一次内部试用发布

内部试用发布是指将你的软件发布给公司内部同事使用。这是紧跟在特性完成之后且在编码完成之前的关键事件。内部试用发布给了你一个展示产品真实进展的机会。它还能为你的团队带来称赞和反馈，这有利于振奋团队士气并确保你的产品走在正轨上。

2. 使其他试用者能够方便地提交 Bug 报告

建立邮件列表收集试用 Bug 是一个很好的监控缺陷数量的方式。如果你的 Bug 跟

踪系统中没有能让试用者快速提交 Bug 的途径，你可以简单地通过在谷歌文档中创建一个在线表单来专门收集 Bug，谷歌文档会帮助你整理并提供关于 Bug 的报告。你可以要求你的测试主管基于这些收集上来的信息撰写 Bug 记录。

3. 软件发布后应继续进行内部试用

亚马逊和谷歌都维护了一套允许内部试用者看到特定特性的实验性框架。这些框架允许某些软件在生产环境中运行但只能被内部试用者看到。对于你的团队来说，投入构建类似的系统是明智的，因为在新特性正式发布前，你需要一段时间来测试系统的稳定性以及发现一些需要试用一段时间才会出现的 Bug。拥有这样一套框架还能允许你一边收集反馈一边完成产品开发，因为内部试用者是在生产系统而不是测试系统中使用产品的。

4. 让进行内部试用成为企业核心价值观

在推行内部试用的过程中诸如拒绝试用或试用了却因为太忙而不去提交 Bug 的现象太普遍了，真为他们感到羞耻。但抱怨于事无补，你最好根据之前提到的建议不断提醒你的同事们去试用。如果试用没有效果，你需要找到问题所在并解决它。在这个时候，你还可以依靠可信测试者们（后面会详细提到他们）。

5.6 如何开展找虫总动员

找虫总动员是指发动你的团队或者你的整个公司专门花一定时间，通常是一个小时，来寻找尽可能多的内部试用产品的 Bug。一次成功的找虫总动员总是能找到一箩筐值得修复的 Bug。以下四件事情有助于找虫总动员获得成功。

- 设立奖项，提供物质激励。T 恤之类的奖品效果很惊人。
- 在项目计划中增加找虫总动员这样一个关键事件。安排好找虫总动员的时间以方便你的整个大团队了解何时可以参与。
- 将找虫总动员排进你的开发和测试日程表中。
- 感谢每一个 Bug。铭记这一点：“坏的消息就是好的消息。”每发现一个坏 Bug 都是好消息。

5.7 准确且有条理地处理 Bug

我经常在电话面试中问那些应聘产品经理的人：“你是如何处理 Bug 的？”他们的回答之疏漏总是让我吃惊。在我看来只需简单的 3 步就能有条不紊地把 Bug 处理好。

1. 基于频率、严重性和解决成本对 Bug 进行分级。

2. 每天与开发主管和测试主管碰一次，评审新增的 Bug。

3. 不断施加压力以减少新的阻碍发布的 Bug 出现。如果不这么做你就永远到达不了零 Bug 率（零 Bug 率是指没有任何阻碍发布的 Bug 再被引入）。如果到达不了零 Bug 率，你就永远发布不了。

首先要对 Bug 进行分级。你的任务是想清楚应该修复哪些 Bug，仅修复非常严重的 Bug 是不够的，那些丑陋且易于修复的 Bug 也应被修复。因此在对一个 Bug 分级时你需考察以下三个方面。

1. 频率

频率是用来衡量 Bug 出现的频繁度的，每十次就有一次出现？还是只在服务重启时出现？又或者每次用户登录时出现？Bug 出现的频率越高，修复它的重要性就越高。

2. 严重性

你需要评估 Bug 对用户体验的伤害有多大。像伤害较大的如安全或隐私漏洞这类 Bug 应归为高严重性 Bug，而拼写错误这类 Bug 则应归为低严重性 Bug，尽管出现这种 Bug 着实有些说不过去。

3. 修复成本

拼写错误这类 Bug 修复成本非常低，而一个关于无法跨服务器共享同一用户会话的 Bug 修复成本则十分高昂，你可能需要砍掉一些特性才能腾出资源来修复它。

当你和你的团队弄清楚该如何对 Bug 进行分级后，就可以进入第二步，每天开一个 Bug 处理碰头会来决定哪些 Bug 需要修复。产品经理、开发主管、测试主管都需参与进来评审这些 Bug。但这个过程的弊端在于讨论清楚每个 Bug 的处理方式所需的时间是个无底洞，而处理Bug的过程又相当无聊，因此你最好尽量压缩这种会议的时间。以下有几种可供参考的方法。

1. 确定通用的 Bug 评判标准

我通常这样来评判 Bug：“我的那帮高中同学遇到这个 Bug 后我会因此羞愧吗？

他们有多少人会遇到这个 Bug？遇到这个 Bug 会对他们产生持久的伤害吗？”在这些问题上你和你的开发主管、测试主管可能会有不同的观点，但通过这些问题你们很快能讨论到一块去。

2. 先处理最严重的 Bug

测试团队提供的 Bug 列表中已经包含了初始的严重性评级，你们可以参照这个评级优先快速处理那些最严重的 Bug。

3. 限定会议时长

如果会议超出时长则将剩下内容安排在第二天继续讨论。这有助于你管理自己的精力。

4. 只围绕频率、严重性和修复成本来讨论

要求测试主管参会的一个原因在于，他能对修复 Bug 的成本发表看法，且还能辨识出一些看起来无伤大雅其实可能引发更深更可怕缺陷的 Bug。在这个场合下注意不要深究问题产生的根源。如果发现某个 Bug 比你想象得更为严重，提高它的严重性等级然后继续讨论下一个 Bug。

5. 讨论每个 Bug 的时间不要超过一分钟

将因描述不清而无法处理的 Bug 重新退给报告人描述清楚。将需深入调查而不能当下处理的 Bug 放入待调查 Bug 列表中。一分钟法能帮助你避免过于细节的讨论。我发现一旦团队适应了这个节奏，所有人都会力图保持下去，因为没人喜欢处理 Bug 这件事。

过了一段时候，你会发现虽然 Bug 数量正不断下降，但新的 Bug 还是不断冒出来。这就是第三步要做的事情：你需要调整 Bug 的评判标准以防止出现阻碍发布的 Bug 出现。作为通用指南，这个方法这似乎有悖常理。毕竟你也不想因为草草发布产品而蒙羞。但事实是你只要写新的代码就会继续产生新的 Bug。如果你想最终达到零 Bug 率，就不得不停止增加 Bug 到发布前必须修复的 Bug 列表中。你应随着发布日期的不断接近而逐步谨慎地提高 Bug 的评判标准。

5.8 发挥可信测试者的作用

可信测试者是指在保密协议的约束下，在产品发布前使用产品内部试用版的用

户。他们比你的团队有着更丰富的多样性，包括更多不一样的电脑，更多不一样的期望，而且他们还不像你们那么懂技术。因此他们的反馈具有更大的价值。

在亚马逊的时候，我拥有一群可信的客户评价写手。我给了他们我的直接联系邮箱，他们会给我提一些非常有用的反馈，而且经常比我的工程团队更早发现产品上的问题。他们也会毫不犹豫地写信给杰夫・贝索斯，每当他们这样做了之后，我就会收到来自杰夫的邮件。作为团队主管，你要是收到了董事长的邮件，就必须放下任何事情，然后全力搞定邮件里提及的问题。

在谷歌的时候，我们拥有上百个企业用户参与 Google Talk 的可信测试者项目。我们向他们开放了我们内部在使用的所有实验性功能，并邀请他们发送反馈。他们给了我们很棒的反馈，并帮助我们正确定位了质量问题。在这个案例中我们的主动参与率约为 15%。

为了让可信测试者们能够帮助我们改善产品，我遵循以下几条最佳实践。

1. 让企业用户签署保密协议并提供正确的联系方式

可信测试者签署的保密协议和雇佣或者业务拓展中签署的保密协议不同，这份保密协议主要用来保证你有权在产品中使用来自客户的改进建议。咨询你的律师以获得关于这份保密协议应包含什么内容的建议。

2. 制作粗略的新手指南文档，其中包括已知问题的列表

使用谷歌站点来集中存放这些资料是个不错的方式，你可以方便地将它分享给任意多个邮件账户，还可以简单快速地更新它的内容。

3. 创建一个包含整个工程团队的邮件组

当你使用这个邮件组来接收用户反馈时，整个团队都能收到用户发送给该邮件组的邮件。我向来坚信工程团队应该尽可能接近用户，这样他们才能感受到他们开发的是一个实实在在的有用户使用的软件，从而变得斗志昂扬。这个邮件组还可以帮助你减轻工作负荷，因为你的工程团队可以帮忙回答用户提出的一些问题。

4. 让这些用户使用和工程师同样版本的试用产品

部分情况下你的工程团队可能使用的是每日构建版本，这时最好不要让可信测试者也使用该版本，因为每日构建版本没有经过测试且改动频繁。

5. 调研可信测试者

你可以使用谷歌电子表单或者 SurveyMonkey 来收集他们对产品质量的大体印

象。你还可以借这个调研顺便了解下他们对定价的看法，毕竟这些用户都真实体验过产品了。

6. 当产品更新时通知可信测试者们

每发送一封最近产品更新内容的邮件给可信测试者后，产品使用量都会有一个小幅度的攀升。因此理想的做法是，产品每更新一次你就发一封通知邮件，这样外部用户测试的覆盖率会增加不少。

5.9 思想火花：以新用户的方式来使用整个产品

在我看来人们常常是被细微的事情所打动。如果你的内部试用搞得好，产品的大部分地方都不会让你蒙羞。但产品开箱体验的好坏取决于产品中一些最复杂的部分。（开箱体验是指“我刚刚打开箱子，里面是什么呢”，而不是指“嗯，让我们现在跳出框框来思考”。）你需要特别关注自己是如何创建账号以及为账号添加信息的。在内部试用时这些任务你可能只执行过一次，而且那次可能还是在四个月前做的。

Pinterest.com 向人们出色地展示了如何创造一个完美的开箱体验。它的注册极其简单，你可以使用 Facebook 账号，它的落地页面也很抓人眼球，里面填满了各种你可能感兴趣的美图，还有建议你关注的会员。

这里有一个小技巧能帮助你准确体验到一个新用户会体验到的东西：抵达特性完成阶段后删掉你所有数据和账号然后从零开始使用软件，抵达编码完成阶段后再这样操作一次。

第 6 章 赢在量化 6

SHIPPING GREATNESS

一个团队的能力怎么样，通常要看他们的量化数据表现怎么样。量化数据是团队主管的命根子，作为团队主管你全部的工作都是说服别人或被别人说服，各种量化数据为这样的讨论提供了理性基础。没有量化数据支撑的表达就像布偶动物发出的叫声。量化数据还可以帮助你优化判断，或者至少为你的判断提供辩护。好的主管能根据量化数据发现问题、跟踪进度并庆祝成功。

6.1 如何采集正确的量化数据且只采集正确的量化数据

有一个也许是虚构的关于菲多利食品公司如何凭借一个指标数据发展壮大的故事。菲多利的食品在商店出售，就需要占用至关重要的货架陈列空间。理想情况下它应该精确地占用它所需要的陈列空间，占多了就得退货，占少了就浪费了销售机会。

你能设想出很多凭借一个量化数据运转业务的方法。你可以每天粗略统计货架上货品的数量然后预测销售趋势，但这极为费时，更何况货品通常每两个礼拜就需进货一次。你也可以估量一下商店的赢利能力，然后检查目前的存货水平，但是这样得到的数据会被商店本身的效率和规模等因素混淆。

菲多利通过测量“过期数量”解决了这个问题。过期数量是指每次补货时该类货品因为过期而需要退回的数量。菲多利希望这个过期数量刚好是 1。也许你觉得每次都要退还商店一包薯片的钱让人难以接受，但作为测量成本而言它是非常低廉的。如果过期数量大于 1，商店就降低存货水平。如果过期数量为 0，他们就提升存货水平。这真是一个既可现场测量又能指导决策的简单至极的量化指标。

从菲多利这个例子中我们可以发现一个优秀的量化指标应具备 5 个关键特点。

1. 测量成本低廉

过期数量被认为是一个极为出色的量化指标的原因在于它无需测量，现成的退款数据就说明了过期数量。

2. 测量可靠且可重复检验

可靠性和可重复性使你的测量经得起检验，这有利于确保指标能够发挥作用。举个例子，你更换了商店的油炸食品存货管理员后，发现商品的过期数量上升了，你可

以以此为依据去调查并判断，这可能是由于管理员之间不同的行为模式而导致的，比如说新的管理员把货架上的油炸食品给摆在内侧了。当然，即便把菲多利家的油炸玉米饼摆在外侧，让消费者能看到它上面的营养信息，也不见得能对销量有什么帮助。

3. 能频繁地测量，最好能实时测量

亚马逊订单跟踪系统是我见过的在这方面做得最好的系统之一。亚马逊有足够的订单和数据，使它可以建立一个关于订单的实时的统计过程控制模型。如果你发布的特性破坏了订单流，不必惊慌，你的程序会立马自动下线。

4. 团队能够根据它做出明智的改变

像菲多利的油炸食品存货管理员可以根据过期数量即时改变存货水平一样，你的团队也需要能在一个量化数据发生变化后知道该做些什么。举个例子，亚马逊的订单测量系统无疑是十分优秀的，但它只是一个反映健康度的指标。它会在订单量高于或低于预测值时发出警告，你能够知道出现问题了，但不知道问题在哪。更进一步说，每个团队都想把订单量升上去，但亚马逊的产品数量太庞大了，任何团队都无法根据全球订单量的变化来衡量他们造成的影响有多大，但如果你们团队是负责购物车的，你可以测量用户从开始结算到购买成功的转化率，这个数字反映了用户体验和系统性能的好坏，并给了你一个可以瞄准的目标。

5. 专注于客户

我喜欢购买转化率这个指标的另一个原因是，它反映了客户的体验状况。一旦你的系统变慢或者你往用户的购买流程里面塞入了太多的步骤，这个指标就会下降。而你的指标如果是后端 Oracle 数据库 99.9% 的平均延迟时间的话，只要延迟没有达到 2 秒或 3 秒，从指标上看不管你做什么好像都不会对客户体验产生影响。所以你的任务应该是尽可能采集那些贴近客户的数据。只有贴近客户的指标才富有意义且可被理解。

专注于客户的另一个要点是尽可能测量客户体验的末端。比如你开发了一个可供下载的 iPhone 软件，那么哪个指标更有意义呢，是下载量，还是软件启动量？我认为是软件启动量，下载量只告诉了你营销做得怎么样，软件启动量才会告诉你用户的参与度和增长情况。

因此你的任务是识别出产品中的“过期数量”。在你打算创建一个复杂的算法来指导业务运转之前，回忆一下第四点，它告诉你和你的团队：一个好的指标要能够指导你们的行动。有些业务仅凭一个数据指标很难运行下去，如果硬要这样做就会催生

出一些没有意义的输出，亚马逊试图将单个数字的“适应性函数”[①]（fitness functions）应用到它的所有团队就是一个例子。当订购团队创造了一个极为出色的可以赖以生存的适应性函数后，其他团队就难受了，他们不得不采取各种复杂的数学变换把之前多个指标转换成一个指标。更夸张的是，为了转换指标，他们投入了非常大的工程资源！这样做下去太危险了。

6.2 你需要采集的三类量化数据

美国管理学大师爱德华兹·戴明曾写道：“无法测量的东西也就无法提升。”显然他是对的。再进一步讲，如果你辛苦了一年去提升某个产品的某些客户的使用周期，但到头来你无法量化业绩，你凭什么能晋升呢？如果你意识到无法晋升，在没有任何拿得出手的业绩的情况下你又凭什么能找到一份新的工作呢？

如果想在未来证明你的业绩，你需要事先准备一根基准线。因此你必须尽早确立指标并在产品开发过程中不断更新。确立基本指标并不困难，比如说工程团队的执行能力就是一个基本指标。

执行力可以通过考察产品能否在你要求的日期内发布来衡量。你的发布时间通常取决于待修复的 Bug 数量。很多 Bug 跟踪系统能够生成发现 / 修复率和 Bug 数量趋势图。因此综合发现 / 修复率和 Bug 数量你可以预测“零 Bug”到达日期。要了解更多有关如何生成该指标数据以及它为什么这么重要的内容，请参考 4.3 节。

零 Bug 到达日期是一个优秀的反映开发进度的指标：它几乎没有获取成本（大部分 Bug 跟踪系统都能免费提供该数据），它可靠且可重复，它能实时更新，它能为团队指明方向。在后面这个例子中，如果你让团队分心去实现你的一些“卓越的想法”，你的发现 / 修复率就会下降，发布就会延期。因此要想尽快发布产品你就应放弃那些“卓越的想法”。

产品发布后你可能需要更换指标，因为你新引入了一个重要的数据源：客户及其行为数据。你需要借助基于它们的指标数据来向投资方或管理层汇报，形成产品发展策略，并指导你的团队。下面列举了三类发布后需要跟踪的关键指标：目标进度、经

① “适应性函数”也叫“评价函数”，是遗传算法中判断群体中个体优劣程度的指标。——译者注

营绩效和系统性能。

6.2.1 目标进度

目标指标会告诉你目标的完成进度。在谷歌一个重要的目标指标是“7 天活跃用户数”。它是指近 7 天使用产品的独立用户数。这个指标比那些廉价的网站分析套装中经典的“日均独立用户数”要靠谱得多，它反映了产品当前的状态，你可以拿它同上周、上上周进行对比。它还比日均用户数更为合理，因为你很少会做一个预期人们会每天使用的产品。

如果你确实在做一个预期人们会每天使用的产品，那么比较单天活跃用户数和 7 天活跃用户数之间的差额也是很有意义的。举个例子，我曾经负责面向 Microsoft Outlook 的 Google 插件，产品全名为面向 Microsoft Outlook 的 Google Apps 同步插件 TM。我们预期除非这个软件没做好，否则使用 Outlook 的客户应该会每天检查他们的邮件。因此我们密切关注 7 天活跃用户数与单天活跃用户数之比。如果你提供的是一个像“照片打印”这类用户不会频繁使用的服务，你可以更关注“30 天活跃用户数”一些。

你还可以跟踪如利润、注册量、下载量或安装量等目标指标。

这个时候你可能想，“目标这个我都懂，不就是满足 SMART 原则嘛！”什么是 SMART 原则？不久前一些“业内砖家”提出目标应该具体、可量化、可到达、合理以及具备时效性。这个框架看起来不错但不够具体，我更喜欢精确增量表达法（Great Delta Convention，第 10 章有详述）。没有人会看不懂使用精确增量表达法来描述的目标，而且这样的描述基本符合 SMART 的定义（除了缺少了“合理”这点）。

6.2.2 经营绩效

经营绩效指标会告诉你产品的问题在哪里以及如何提升用户体验。这些指标通常是用比率表示，比如从点击购买按钮到付款成功的转化率。和目标指标一样，选择合适的经营绩效指标至关重要。比如说你想做一个优秀的社交产品，监控好友数量是没有用的，不同类型的用户有不同的好友数。你应该去监控用户参与度，这样你才能回答“用户会花费时间在网站上吗”、“用户会发信息吗”这类问题。反映这些行为的相关指标可能包括 7 天活跃用户平均 7 天发帖量、7 天活跃用户平均停留时间等。

埃里克·莱斯在他的《精益创业》一书中并不推崇那些总是在增长的指标。他称它们为虚荣指标，因为即使新用户的流失率高达 90%，你也可以昂首挺胸指着一路上涨的图表向大家宣布：“我们太了不起了，我们的产品正在成长！”莱斯的观点很客观。这也是为什么你要关注转化率、参与度等指标的原因。几乎所有的网站数据分析套装中都会直接提供转化率相关的指标，它们还会告诉你哪组用户使用过哪些特性以及点击过哪些按钮。

另一个避免“虚荣”指标的方法是衡量应用中改动带来的影响。最好能让改动前和改动后两个版本横向而不是纵向对比测试，因为纵向的分析常常会被其他因素影响以至于你可以轻松地宣称：“看，我们的产品用户仍在增长！”Google 分析提供了极其强大的 A/B 对比工具，不过这只是很多有用的工具中的一种。大多数主流网站都有一套支持灰度发布特性的测试框架，它能确保新的特性或者体验达到预期的效果。因此只要有一丝可能你都应该尝试在项目一开始就构建一套实验性的框架（参考第 7 章关于实验性发布的其他好处的讨论）。

6.2.3 系统性能

系统性能指标能说明你产品的实时健康度。这类指标包括 99.9% 平均延迟、每秒请求数、并发用户数、每秒下单数以及其他基于时间的指标。如果这些指标大幅下降，那么一定有什么地方出了问题，你也许需要下线一些程序。

你还可以天马行空地从统计过程控制（SPC，Statistical Process Control）的视角来考察你的指标。爱德华兹·戴明是最早推广使用 SPC 来精确测量一个指标的正常下降范围的人之一。他在 20 世纪 20 年代从沃尔特·休哈特那里学到了这个东西。戴明假设你的系统存在一定的波动，伴随着波动存在一个可以接受的系统变化范围。在这个范围内的波动被认为是正常变化或 I 型误差。

系统还会有一些持续一小段时间的不良的尖峰波动，戴明称它为异常变化或 II 型误差。一次错误的推送或者一个丢失了虚拟 IP（VIP）的服务器可能会引发这样的尖峰波动。

你可以忽略正常误差或正常波动，即误差或波动落在平均值的正负标准误差区间内。标准误差是指你的数据的平均值的标准偏差。如果单个数据点落在平均值的正负标准误差区间之外，你就得呼叫技术主管了。

6.3 专注于目标本身，忽略细枝末节

几乎所有的指标都可以通过一些巧妙的手法进行操纵。以之前举过的发布日期这个例子为例，为了将发布日期提前，我们可以将更多 Bug 归到“不影响发布”这一类去，也可以简单地终止测试（表面上看还是一个“双赢”的做法）。但实际上你和你的团队是不可能糊弄系统的，指标只是一个指示器，不是你的老板，所以请放心，你的核心指标是不可能被糊弄过去的。当指标变成了你的老板，你需要花费数天乃至数周的时间去为你指标数值的合理性辩护时，你就该换个指标了，或者换个工作也行。

第 7 章

赢在发布

7

SHIPPING GREATNESS

恭喜你！你的产品 Bug 已经（几乎）全部修复了，可信测试者很喜欢它，团队成员也颇自豪。你还构建了一套衡量产品成败的指标监控系统，甚至连产品发布通告都有了一个初版（你在第一次定义产品时写的）。

万事俱备，只欠发布，不过发布可不只是把文件上传到服务器。这里有几个主要的发布步骤，遵循它们可以确保发布质量。

1. 对改动说不。
2. 开启作战室。
3. 营造紧迫的气氛。
4. 核查发布清单。
5. 撰写博文。
6. 发布软件。
7. 亲自验证软件。
8. 应对发布带来的各种影响。

7.1　对改动说不

在准备发布的过程中你必须尽量频繁地对新的特性、新的 Bug 以及用户体验上新的改动说不！如果不这样做，你就永远完成不了软件，自然也就永远交付不了。软件业中有句格言："发布手中有的，而非脑中想的。"这句格言之所以被业内人士常常提起，是因为它真实反映了现状——有时候你不得不交付你的产品，即使它并不完美，因为交付一个过得去的产品比为了追求完美而什么也交付不了好。大部分人都认同这个观点，但真要执行起来却发现困难重重，因为产品做到什么程度才算"过得去"并没有明确的衡量标准。

为了减少这种衡量的随意性，我会通过检查团队是否对目前的产品抱有自豪感来确定产品是否可以发布。你的团队必须对他们开发的软件抱有自豪感，残存在产品中的 Bug 也不应该让你觉得羞耻。除了这些告诫，你还需要敢于坚守你在数周及数月之前做出的决策。这里有一个方法可以让你的团队愿意说不，即建立一个"发布后第一

时间”（IPL，Immediately Post-Launch）修改的需求清单。你需要让你的团队认识到有些改动固然重要，但它不一定要在发布前完成，你们可以在发布成功后第一时间进行改动，这样团队会对产品更有信心，他们知道他们顾虑的东西将很快得到解决。

另一个你应该对新冒出来的改动说不的主要原因是，几乎所有对代码的改动——更合适的叫法为代码移动，都会引发新的风险，如引入新的 Bug 或重新引入老的 Bug。重新打开老的 Bug，或者之前运行正常的功能运行失败，都被称为回归 Bug。为了避免回归 Bug 并推动开发持续前进，团队通常会创建一个发布分支。发布分支是你打算交付的软件版本，你之后只会给这个分支添加一些用于修复关键问题的代码。新特性的开发可以在开发分支上进行。这个方法固然有效，但它会增加管理成本，因为工程团队必须同时维护两个分支。不过只要工程团队的工作量没有翻倍，这个额外的成本还是可以接受的，所以在涉及定义分支时你应尊重开发主管的意见。

我会尽可能推动工程团队去维护一套发布分支，原因很简单，它能帮助我的团队应对危机。因为发布分支在发布后不会有任何变更，当出现安全、隐私或重大性能问题时，你便能基于发布分支开发一个“热修复补丁”，而且由于没有其他的改动，需要执行的测试量也不大。因为改动和测试都能很快完成，所以在发现问题后你能在极短时间内发布热修复补丁。

7.2 开启作战室

随着发布日期的临近，你的会议安排必须进行调整。每周开一次会的节奏已经不合时宜了，所有人都在朝着目标加速冲刺。在这个节点上你应改开每日例会并不再禁止与会者在会上争论一些问题。因为这时候效率是最重要的，有些问题最好能在会上马上解决掉，尽管这问题可能与半数参会人无关。每日例会能帮助你高效做出决策并营造一种紧迫的氛围。

因为这类会议牵涉面有点广，它变得越来越像个作战室，身处其中的人们紧密合作以处理各种新冒出的问题。在软件发布的实际过程中系统由于过于复杂而常常会引发一些令人烦躁的小毛病，因此有时候开启作战室能促进团队的协作并消除信息的传递时间。试想如果需要沟通的两个人坐在一起，那么他们之间的信息传递时间实际上为零。

7.3　营造紧迫的气氛

有一个颇有争议的观点认为，项目管理不当是造成项目后期团队紧迫感加剧的原因。看起来的确是这样，只要组织良好，计划得当，项目就能按部就班地完成发布。但实际的软件项目并不遵循这个理论，所有的项目都是看似时间分配得井井有条，但到最后都需要冲刺一把才能赶上发布时间。我还没见过例外的情况，所以接受这个事实吧，切记别人急躁之时正是你保持冷静之刻。

我认为凭借冲刺来完成项目没有什么不好。只要这样的冲刺不超过 1 个月，大多数团队和他们的家人还是可以接受的，特别是你还会补偿给他们一定的休息时间。如果你在距发布还很遥远的时候就发起冲刺，发布后还不结束，那么你导演的就是一场死亡行军。死亡行军是一场灾难。它会摧残你的团队，催生劣质的软件，而且没有任何趣味可言。所以不要让你的团队冲刺超过 1 个月，或者最多 2 个月。如果你发现老板要求的发布日期不切实际，你必须咬紧牙关将这个信息告诉你的老板，尽管你知道你最终会被塞上一堆狗屎三明治[①]后无功而返（参见第 12 章），但务必在团队冲刺前将这个信息告诉老板！

如果确定要冲刺了，那么在最后冲刺期间你最需要做的事情便是确保你的相关方和支持团队都感受到了这种紧迫的气氛。新手工程师通常会羞于督促其他团队或向别人请求帮助。他们害怕别人发现他们有不懂的地方。这样的沉默会造成心理问题，因为承认自己有所不知是心理走向成熟的标志，往往第一个承认自己有所不知的人都是知识渊博的人。让你的开发主管特别关心下团队中的新手工程师吧。

有一个方法可以将紧迫氛围渗透到你的从属方，即召集他们团队和你们团队在电话会议上进行讨论，你在其中进行斡旋。这种讨论更像是一场谈判，你要让双方先就目标达成一致，然后再讨论出一个创造性的解决方案（参见第 11 章关于谈判的小贴士）。你还需要记住你的任务不是去解决问题，而是创造出一个有利于问题解决的环境。当所有人都被召集到电话会议中后，确保他们说的是同样的语言，并在过程中将你的紧迫感传递给相关方。

---

①狗屎三明治（The Shit Sandwich）是一种有效的反馈技巧，出自于《一分钟经理人》（*The One Minute Manager*），它是说当你给员工反馈时，你可以一开始先表扬他们（第一片面包），然后再给他们一些难以完成的挑战（狗屎夹心），最后提醒他们你有多么看好他们的能力（第二片面包）。——译者注

7.4　完成发布清单的核查

要想出色地完成发布，你需要拟定一张发布清单。这份清单的目的在于确保软件发布中所有需要跟进的事项都被有序安排且被详细描述。发布清单还能促进团队内部不同职能的交流。正确地使用清单能让它发挥不可思议的作用，每个民航飞行员在每次飞行前都必须核查一遍清单，可见清单的价值和重要性。

你的每个主管，从设计到测试，在清单中都列出了各自需要负责的事项。完美的可适用于任何项目的清单是不存在的（就像每架飞机都有它独特的核查清单），每个子团队，如法务、测试或市场，都可能有着各自的子清单。但你应尽量把需要跟进的事项加到清单中。图 7-1 展示了一个简单的清单示例，可以在 http://www.shippinggreatness.com 下载到。

	A	B	C	D	E	F
3	领域	事项	负责人	状态	预期完成时间	参考信息
4	工程					
5		找虫总动员	克里斯，产品	完成	7/15/2011	http://bugzilla/123
6		指标监控中心	克里斯，产品	完成	8/1/2011	http://www/dash
7		负载规划，负载	萨莉，工程师	进行中	8/1/2011	
8		零BUG率	拉里，测试	打开	8/10/2011	
9		切出分布分支	萨莉，工程师	完成	8/15/2011	build 0731
10		安全评审	萨莉，工程师	完成	8/1/2011	
11						
12	运营					
13		通知所有依赖的服务	克里斯，产品	打开	8/10/2011	
14		轮班安排	萨莉，工程师	完成	8/20/2011	http://www/oncall
15		商标评审	埃德，法务	打开	8/1/2011	
16		专利评审	埃德，法务	进行中	8/1/2011	
17		隐私评审	埃德，法务	打开	8/1/2011	
18		博客文章	马丁，公关	进行中	8/15/2011	
19						
20	发布当天					
21		推送至生产环境	萨莉，产品	打开	8/25/2011	
22		推送后验证	拉里，测试	打开	8/25/2011	
23		通过实验性框架发布给1%的用户	萨莉，工程师	打开	8/25/2011	
24		通过实验性框架发布给10%的用户	萨莉，工程师	打开	8/25/2011	
25		通过实验性框架发布给100%的用户	萨莉，工程师	打开	8/25/2011	
26		发布博文	克里斯，产品	打开	8/25/2011	

图 7-1　发布清单

7.5　撰写博文

如果你是完全按照产品开发过程行事的，那么你应该已经写过一篇博文了，或者至少写过相同主题的产品预告。那篇博文的目的在于阐述你的使命、你的目标客户以

及你能解决的问题。从传统新闻的角度来看它就是你的“导语”。下面列举一篇范例，它截取自 Google+ 发布博文的开始部分。

与他人建立连接是人类最本质的需求，我们每天都在通过各种笑声、低语或者欢呼来与他人建立连接。

今天，人与人之间的连接越来越多发生在线上。然而那些在真实世界中互动的微妙之处和本质所在却因我们僵化的在线工具而消失殆尽。

在这个背景下，本应顺畅的人类分享行为在线上也尽显笨拙，甚至无法进行。而我们的目标就是修复它们。

谷歌瞄准了一个清晰的群体：那些愿意在线分享的人。谷歌还发现了一个问题：当前的在线分享是“僵化”的、“笨拙”的。在接下来的篇幅里，谷歌深入介绍了解决这种笨拙的四个独特方法。其实 Google+ 与现有的社交工具之间没有那么大的差别，但谷歌却能把文章做得花团锦簇，所以你也应该大胆发挥你的想象力！

你的任务在于构建一篇同等质量的博文。你可以从产品预告中挑选有用的文字，如果仍然需要使用具体案例则需抛开细节。你还可以邀请市场团队帮你润色。总之撰写博文是一个你可大展身手的机会，但你应始终保持公司的口吻。

博文还是你产品演示的脚本。你可以准备一个一分半钟到三分钟之间的视频放在博文后面，这样那些没有权限或者对试用缺乏耐心的用户仍然能体验到你产品的卓越之处。

7.6 发布软件

发布特性的最佳方法是借助一套实验性框架。它允许新旧两套代码同时在产品服务器上运行，这样无需重启服务器即可在版本 1 和 2 之间快速切换。长期来看，投入资源构建一套实验性框架几乎总是值得的。

如果你仔细观察过谷歌和亚马逊等主流网站改版，你会注意到有些用户体验到的东西和其他人不一样，这是因为这些网站都使用了实验性框架来部署和测试特性。亚马逊称这样的发布为网页实验室（Weblabs），因为他们会去比较拥有新特性的用户和未拥有新特性的用户之间的差异，进而衡量新特性对用户产生的影响。是的，你的某些用户会面临与其他大多数用户不同的体验，但这没有关系。过往事实证明用户不会

介意这些。如果他们真要介意的话，回滚软件然后庆幸你之前做好了回滚的准备吧。

你可以使用谷歌分析提供的免费系统来构建类似网页实验室式的发布。我曾经使用过它，效果很不错。当时我领导的一个团队借助谷歌分析进行了一系列实验，我们成功将转化率提升了 65%——这个数据也是谷歌分析告诉我们的。

然而实施这套实验性的方法还是面临很多挑战。当你的服务需要改动底层数据模型时，你可能没有办法回滚，或者回滚之后会产生一些不相交的数据集。这些问题都很棘手，因此有可能的话你最好让你的数据模型向后兼容，这样可以彻底避免这些问题。我知道这个建议有点站着说话不腰疼，但如果真能做到在不丢失用户数据的前提下回滚，你将收益巨大。

再多的努力也很难让你做到一次性发布成功。因此建立一套机制来确保所有东西在发布前就已衔接完成并配置完好是极具价值的，这样你才不会把问题暴露给上百万的用户。

不要选择在周五或者临近假期发布。先不说你会因此错过放松的机会，想必你也不想为了推出产品更新包而花整个周末的时间去拼命呼叫工程师们吧。我曾经愚蠢地同意在 12 月 18 日推出一些软件来促进假期 PC 的购买，结果上线后不久服务出了故障，程序被直接下线，于是我不得不在平安夜里呼叫那些正在全国各地享用晚餐的工程师们。这不是一个笑话。

7.7 亲自验证软件

在将所有服务推送至生产环境后，你需要在 2 天内完成产品验证。首先，你的测试主管会精心组织一轮验证，这个验证也被称为版本验证测试（BVT，Build Verification Test），一般在新版本发布后进行。通过这轮验证，测试主管能确保推送到生产服务器上的版本是正确的，并且所有的配置文件都已推送成功并安装正确。这个环节容易漏掉一些简单的小细节，如域名中只配置了 http://www.domain.com，而没配置 http://domain.com。

其次，你需要以新用户的身份来亲身体验整个产品，确保产品所有主要功能都可正常使用。有些产品功能常常会出现问题，如注册流程、上传数据（如图片）、搜索、表单提交等。它们都依赖于一些子系统，所以有时会因为配置疏忽而指向到了错误的

服务器。这种类型的错误无时无刻不在发生。因此你的团队应该等待你和你的测试主管、开发主管全部验收通过后，再把产品推向更大面积的用户。这也许看起来步骤很多，但其实只需要你们三个在即时通讯软件上沟通确认一下即可，所以不会产生多大的操作成本。

7.8　应对发布带来的各种影响

如果你验收通过了产品，恭喜你。你做得很棒，你的产品也终于发布了！但交付还没有完成，你还有些事情需要做，比如处理可能涌现出来的危机，向全世界宣传产品，以及和团队共同庆祝。下面列出五项发布后需要做的事情。

1. 应对回滚。
2. 处理产品危机。
3. 演示产品。
4. 应对媒体。
5. 庆祝发布。

7.8.1　出现问题，回滚软件

这句话值得反复强调：*只要成功回滚，发布就还没有失败*。回滚是指把软件撤回到预发布状态。它简直就是家常便饭。我在亚马逊和谷歌都见到过一些发布回滚了 5 次甚至更多。大型软件系统的接口和依赖太多太复杂了，你几乎不可能在发布前测试和验证完所有可能性。如果发生回滚，只要对用户没有产生明显伤害，你就还没有失败。所以最终你会发现：最好的防守是制定周详的撤退计划。

有时候你可能无法回滚或因为回滚成本太高而不愿回滚，这种情况下你的最佳选择是确保你的团队能在发布后数天内继续高歌猛进，因为你们需要一段时间来找到问题、修复问题，同时在修复问题的过程中，你的客户还得继续忍受这些问题。所以从另一方面讲，如果可以回滚，你就能撤回对产品的改动，从容不迫地修复问题，然后再试一次。

7.8.2 应对产品危机

危机有时会突然爆发出来，可能是你的网站被瞬间大规模的流量冲垮，也可能是出现安全漏洞、隐私侵害或者定价事故，又可能是一个实习生把本应重定向到数据中心的产品站点重定向到他的个人桌面了（真实故事）。像这些情况你都可以参照下面的这份可靠的行动计划来应对。同时和所有优秀的应对措施一样，下面这份行动计划也受到童军运动[①]的启发：做事之前，先做准备！

先做准备部分意味着你需要事先安排好人员轮班待命并给他们配备寻呼机。我仍然认为手机不如寻呼机可靠，但我没有找到一个可靠的方法来强制工程师在公司之外也随身携带寻呼机。因此你需要把团队里每个工程师的手机号码都记在手机或者号码簿里。

一个准备充足的产品应该拥有可以快捷关闭服务或限制服务速率的软件开关。切记只要有可能就应该在实验性框架下发布软件，或者软件中带有可禁用特性的标记。记住，只要成功回滚，发布就还没有失败。

你还可以更早做好应对灾难的准备。在早期软件设计评审会议中，你就可以要求工程团队针对可能的错误设计应对方案。比如建立一套逻辑来限制对过载服务器的请求。你需要呈指数级地增加退避（backoff）时间[②]，并总是随机修改它的值，这样退避的行为才不会造成更深层次的破坏。退避的随机修改机制可不是闹着玩的，我曾见到过被不完善的退避机制拖垮的系统比被原始问题拖垮的还多。确保在交付之前建好退避机制。

你还可以准备得更好一些，比如取得某个服务器专家的联络方式，哪怕你的团队里没有这个角色。如果能维护一套紧急联络簿并牢记于心，并在内部公开那自然更好。你还可以创建一个内部 Wiki 来存放公关、法务和跨职能团队的联络方式，这样在找人方面你就基本不会有什么大问题了。当然你也可能遇到一些小的令人沮丧的事情，比如当发现某个特定民族的用户因为莫名的原因突然无法正常使用你的产品时，你可能不知道该找哪个公关的同事帮忙应对。你猜的没错，它在我的身上发生过。

---

①可参考维基百科“童军”条目。——译者注

②退避时间是指网络节点在向服务器发送请求时产生冲突后再次发送请求需等待的时间，如果每次退避时间都一样，那么之前冲突的请求又会再次冲突。——译者注

你还可以使用另一种方法来提前建立有效的沟通渠道：创建一个危机扩大邮件组（-escalation@yourcompany.com）以便所有相关人员都能被纳入进来。他们包括公关、客户支持、你自己、你的工程主管、你的产品主管以及重要的跨职能团队负责人等。把团队的邮件组也添加到这个邮件组中，因为危机是最好的教学材料。

如果存在持续的风险（例如一次发布），你需要确保所有相关人员都知道这个风险可能引发的问题。你必须乐于劳烦别人以获得帮助，要知道如果遇上了大危机你是没有时间去遮掩的。

你需要准备的最后一件事情是把下面的内容打印出来，钉在墙上，然后在大麻烦来临之际按照这个剧本行事。

危机应对“剧本”：0 ～ 5 分钟

■不要惊慌。

这点说起来容易做起来难。如果你的老板打电话过来，十有八九他也慌了。看起来慌不慌和老板薪水有多高或者经验有多丰富没什么关系。尽快挂掉他的电话，然后平复心情。

■检查这是否是一起突发事件并评估影响范围。

你需要评估有多大比例的用户受到了影响，这个影响有多严重。幸运的话这也许只是个小事件。但既然你已经在阅读这个“剧本”，而且你又是搞软件的，可见这不太可能是个小事件，继续看下去吧。

■确定这个问题不止在你这里出现。

有时候你的电脑或账户会以一种奇怪的状态中止工作。你肯定不想上蹿下跳，火急火燎地找人解决后发现唯一的问题只是因为你清空了 cookies。你还需要确定这个问题不止在公司内部用户或内部试用版的用户身上出现。除了使用外部账号验证外，你还可能需要看一下是否有客户在论坛、Twitter、eHarmony 等任何地方发表过相关的信息。你要找到一个确实存在该问题的客户。

不要逃避大问题，出现了就是出现了。不要拼命去欺骗自己说这不是个大问题，没有多少用户受到影响。欺骗自己一切都会变好，只会让真实情况越发不可收拾。即便意识到你离被炒不远了，你也可以保持淡定。确实，这听起来不可思议。不过要是做不到，你就真的会被炒了。

■发起电话会议。

在主持电话会议的过程中，请不要去掺和讨论解决方案。我再次强调这一点。虽然对你来说忍住不去讨论解决方案很难，毕竟你对系统非常了解，而且你还非常关心客户。但是，你在电话会议中的任务是去协调各方对话，促使他们讨论出一个解决方案来，而不是你亲自上阵。如果你也掺合进去，讨论只会更乱。我知道这听起来有点不合常理，但很多管理者都吃过这样的亏。如果你想吃一堑了再长一智的话，尽管去吃吧……

■打开一个 Bug。

你将使用这个 Bug 来记录对系统做出的改动。工程师会把获得的日志片段添加到 Bug 中。任何想收听技术讨论的人都可以将自己的名字添加到 Bug 中。这个 Bug 对你撰写事后调查报告非常有用，因为它将包含各时间戳以及关于你做过的正确或错误的事情的完好记录。

■知会危机扩大邮件组成员。

发邮件给你之前建好的邮件列表，邀请能够阻止危机扩大的第一人一起参与应对危机，无论他是公关、工程师还是网络运维中心的工作人员。确保你收到了他的肯定答复，无论是用电话、短信还是邮件。只在语音信箱里留个言是是不够的。

危机应对“剧本”：5 ～ 30 分钟

■问：“我们能回滚吗？”

解决大型危机的最佳方法是撤销任何引发危机的改动。“前滚”（Rolling forward）需要更多编码，更多测试，因此需要更多时间。而且在重重压力下编码的效率会十分低下，所以你应该首先尝试“回滚”。

■推迟任何公关计划。

在发布过程中你会经常性地（但不是 100%）遇到问题。如果你之前有一些通过市场推广、公关或者其他方式为你目前出问题的产品吸引关注的计划，确保它们已被暂时中止。

■知会相关方。

不要断定危机只对你有影响，确保危机没有危害到其他服务方。如果你危害到了其他服务方，赶紧告诉他们以便他们展开自检。要知道他们也可能有自己的市场推广计划。同样地，你也需要确保你没有被其他服务方牵连。一些重要的共用系统，如你的后端存储，会偶尔因为其他服务方的缘故而出现故障并严重损坏。一旦遇到这种情

况，你是什么都不用做，也什么都做不了，你只能把自己添加到他们的 Bug 中（假设他们也是依照本书所述的过程开发产品的），然后密切观察他们的解决进度。

■ 知会社区。

如果你有一个社区论坛或者其他常用的与客户交流的途径，你也许想通过它们让客户知道“目前某某功能看起来出了一点问题，我们正在积极研究如何解决，预计将在某某时间提供更多信息或解决方案”。Google 应用状态面板（http://www.google.com/appstatus）就是一个很好的例子。让你的公关 / 客户支持团队帮助你起草公告，这样可以避免泄露敏感信息或者制造过度恐慌。然而不要避讳承认产品存在问题，用户会欣赏你的诚实。

■ 保持 Bug 的更新。

记住，其他团队的人们正在关注 Bug 的最新进展，工程师也在关注 Bug 以获得更多背景信息。所以不要把 Bug 的最新情况藏着掖着，更新出来不会对解决危机有什么负面影响。

■ 寻找并引入专家协助团队解决问题。

有些问题很难解决。你的团队如果有充足的时间也许能够解决它们，但如果这些问题正在线上持续地影响着用户，你就不会愿意等待团队去解决它们了。这时候可以考虑引进其他团队的专家来帮忙。当你决定这样做的时候一定要谨慎，你肯定不想因此打击到你的工程团队或者扰乱他们的安排吧，不过一个经验丰富的专家通常知道该如何与陷入困境的团队打交道。

■ 知会管理层。

你之前已经知会过危机扩大邮件组成员了，因此这个时候你的团队都了解目前发生了什么事情。不过你的老板还不了解，但是他们需要了解，因为他们不想在收到他们的老板转过来的用户投诉邮件时茫然不知所措。因此你需要将你老板的电子邮件添加到地址栏，拷贝下面内容到邮件正文并填写完整，然后点击发送。

> 各位团队成员，
>
> 我们正在积极解决关于________（名词）的一个问题，这意味着________(2 句描述用户痛点的话)。

问题是在________（时间）发生的，我们预计会有________（用户数量）受到影响。

我们预计此次问题会导致________________（多少数据丢失或者多少收益损失等）。

目前________（工程师）和________（电话会议组织者）负责确定解决方案。他们下一步将________（动作）

我们预计问题将在________（时间）修复，除此之外，我们会先在________（时间，小于两小时）发布一次更新。

这是关于该问题的跟踪 Bug：________（Bug 编号），更多信息请直接查看 Bug。另外，我们将召开电话会议讨论该问题，你们可以通过这些方式加入：________________（IRC 频道或者电话会议号码）

________________（你的昵称），

________________（你的正式职位和名字）

危机处理“剧本”：31 分钟及以后

有时候危机会持续数小时以上。这时候你需要宽慰你的副手：“痛苦是短暂的，一切都会好起来的。”我在亚马逊的第一个老板就是这样和我说的。在你说完这样一句自豪的、无畏的、充满管理智慧的话后，你需要立即进入持久战模式并继续修复问题。此时不存在任何快速而简单的工作清单，相反，这时的工作都需要你日复一日坚持去做，如下。

- ■定期发送状态更新，当有人请求时也可马上发送。如果有人来追问你状态的变化，那就说明发送的频率还不够高。按时递交状态报告有助于老板相信问题正在被妥善解决中。
- ■不要把客户晾在一边。控制客户对问题的期望，并保持与他们的沟通。尝试在兑现承诺的基础上再多提供一些优化，像这样能够取悦客户的机会是不多的。
- ■继续解决问题。长时间从事一件事情会自然而然地使人们感到疲惫，更何况这还是一个救火的项目。因此如果不营造出一种紧迫感，团队就会失去焦点。
- ■确保满足从事问题解决的人们的需求，提供给他们食物、服务器以及其他团队

的支持等。

- 建立轮班制度，不要让一个开发者持续工作24小时。在亚马逊发生的一次危机中一个工程师对我说："凌晨3点了，我觉得现在的我已经写不出来没有Bug的代码了。"我很欣赏他的这种诚实。
- 采用变通或应急方案。具体来说，就是临时找个方法来替代目前已经瘫痪的系统。比如在谷歌有一次我们碰到了下载服务器性能不足的问题，于是我们把下载流量分到到阿卡迈[①]（Akamai）的内容分发网络（CDN，Content Delivery Network），这样问题就暂时解决了。

危机处理"剧本"：收尾以及撰写事故调查报告

自我表扬一下吧，你总算把问题修复了，老板似乎也没打算开掉你，当然，等他明天看到你发出的事故调查报告后就说不一定了。现在你可以从你的小U盘里拷几首歌到电脑上听听，然后继续下面的收尾工作。

1. 检查修复情况。你需要再三确认问题是否修复完成了，不要在你的团队被质疑时口口声声说"我们100%确定已经修复了问题"，结果却发现你只能"99%确定"。你需要亲自验证修复情况，就像在发布前亲自验证软件一样。
2. 如果你或者你的公关团队认为有必要对外公布此次危机情况，那么准备一篇博客文章。
3. 在团队路线图中增加任务项并把他们的进展同步给老板或投资人。
4. 撰写事故调查报告。

事故调查报告是发给你的管理层的一份基于数据的检讨。写好它很容易，因为它的结构就和小学三年级写的作文一样：什么事情，什么人，什么时间，为什么以及怎么做。之所以按这样的顺序排列问题是因为高管也常常按照这个顺序来提问。下面列举你老板可能会问的问题。

发生了什么事？

列出简要的时间表来回答这个问题，包括问题第一次出现的时间、被发现的时间、团队介入的时间、修复完成的时间以及其他任何相关时间表。

---

①国际上最大的CDN服务商。——译者注

谁受到了影响?

尽可能详细地描述受到影响的客户，包括他们的数量，他们所属的类别以及其他你知道的关于他们的任何有用的信息。

问题是何时出现，又何时终结了?

你应该提供这场危机的基本时间线。

为什么会发生这个事情?

你需要在这段解释事情发生的根本原因。如果你还没有找到根本原因，继续追问“为什么”，直到你找到它（参见第10章关于鱼骨图的讨论）。我在后面的示例中使用了讨论的方式来阐述根本原因，你不一定要像我这样做，但这么做有利于解释为什么你会得出那样的结论。

如何防止这类问题再次发生?

如果你彻底解决了这类问题，那自然最好（虽然可能性不大）。不过既然你还在写事故调查报告，说明你的团队还有很多做法值得改善，你需要让所有人都行动起来。确保每一个人都负责改善一些东西，从而使更多的改善能够落实。

下面是一个事故调查报告的示例，事故调查报告也被称为“故障原因”（COE，Cause Of Error）报告。

COE 报告示例

COE #1 – 令人羞愧的 SQL 注入漏洞。

03/07/12 —— 初稿 —— 克里斯 · 范德 · 梅（cvandermery@）

跟踪 Bug：http://bugzilla/b=1234

什么问题? 广告优化团队发布了一个针对优化程序前端的更新，但在更新中他们没有正确清理搜索语句。同时，数据库运维团队更新了数据库并重写了一些存储过程，但他们没有正确地防范来自 SQL 注入的攻击。一位内部同事在开发入门示例时发现了该问题。

谁受到了影响? 客户在体验过程中感知不到这个漏洞。我们分析了 SQL 事务日志也没有发现任何不规范的 INSERT/UPDATE/DELETE/SELECT…INTO 语句，因此我们认为目前还没有客户在利用这个漏洞。我们也没有收到任何关于此次危机的客户问题报告。

潜在风险：该漏洞暴露给了约 10% 的用户基数，但由于登录用户才能利用该漏洞，所以风险还会再小一些。

这个问题什么时候发生的?

问题发生：5/1/08 14:00

问题发现：5/5/08 15:00

回滚服务器到最近一次正常状态：5/5/08 16:43

问题解决（推送了新的前端）：5/6/08 16:00

为什么会发生这个问题?

我们没有对 SQL 注入写单元测试。

为什么？实际上我们无法让构建运行在有 SQL 服务器的环境里。

为什么？我们无法模拟 SQL 服务器。

为什么？**我们的基本测试矩阵里有漏洞。**

我们无法协调数据库（DataBase，简称 DB）运维团队。

为什么？为了增加自主性，DB 运维专门从我们前端团队中分离出去了。

为什么？团队讨论效率太低了。

为什么？每个人都有不同的观点，我们无法做出决策。

为什么？**没有确定谁拥有查询安全性问题的决定权并承担相应责任。**

如何避免这类问题再次发生？ Cvandermey@：编写单元测试，确保 SQL 注入失败。Harry_the_db_lead@：拟定预部署清单，要求每个需要 DB 运维协助的团队的产品主管验收确认。所有 DB 发布候选版本都要执行清单中提及的测试。所有的测试主管：强调代码评审（code review）的重要性。Charlie_tl@：尤其是要拟定一个代码评审中需检查事项的清单。

7.8.3 演示产品

重新回到有趣的事情上来，你的发布正顺利进行，现在需要你投入精力的是通过一次演示来将产品展现给世人。你的演示需要直截了当，演示的目的在于用讲故事的方式来讲述产品，并在每一步凸显产品使命。它必须简洁，最好不要超过 10 分钟，这样才能保持观众的注意力。如果你把演示制成视频放在博客中，则这个视频决不能

是 10 分钟这种长度的，它需要被压缩在 90 秒以内，否则访客会很快失去注意力。

和撰写博客文章一样，演示也需要遵循既定的策略。先从问题和使命讲起。不断陈述使命，也许你会觉得啰嗦，但这招很管用。在演示开始时一定要让用户明白他为什么应该关心这个产品。

接下来以讲故事的方式把演示串起来。好的演讲总是利用一些故事来吸引听众并让他们产生代入感（第 10 章有更多关于演讲的内容）。你的演示也应以一个贯穿始末的客户故事为主线。用“想象一下”或“人们常会碰到一个问题”这样的句子来开讲故事，观众会很快被你勾住。在演示的过程中不要忧心那些你想要展示的细节。如果出现一些小问题，跳过它，你还可以趁大家注意力分散的时候重申你的核心使命。

不过固然你可以跳过这些小缺陷，但对付它们最好的方法是不让它们出现。所以那些做得最好的演示需要花费数周时间进行准备。比如乔布斯，他对排练的要求之苛刻是出了名的，而且演示要与苹果自家的 Keynote 放映出来的演示幻灯片相得益彰。除此之外，苹果总是在想一些能减少演示过程的差错的招数。

微软的产品经理们也在这么做。他们通常会带上三个笔记本：他们个人的笔记本、用于演示的笔记本以及备用的演示笔记本。这样的多候补策略确保了演示环境的稳定性。我个人也有一些心得。当我需要在线演示一些东西时，我会带上两个不同无线服务商的热点并且总是尽可能使用有线网络。

尽管给你列举了很多前车之鉴，但如果你要做一场全程直播的产品演示，那么再多的准备都不为过。你必须为每一段产品演示都准备截图或者视频，因为你永远预测不出所有可能出错的情况。Google TV 的发布会就由于整个演示过程频繁出现技术问题而惨不忍睹。相比之下，虽然苹果 iPad 2 发布会也出现了类似的问题，但现场马上换成了截图演示，同时工作人员强制关闭了所有与会者接入 WiFi 的权限，最终问题得到了补救。因此你应该像苹果一样准备充分，而不是像谷歌一样草草了事。

7.8.4 应对媒体和客户

如果你有幸能和媒体或者知名博主接触（切记那些博主喜欢别人也把他们当做媒体），尽可能让他们对你的业务产生深刻印象。和他们通电话并向他们演示产品。快速响应撰稿人的需求，因为他们通常都有要求的截稿时间。如果撰稿人发表的文章存在一些错误，你应该谦逊有礼地指出来，撰稿人也许会发布一个更正申明。然而错误

内容一旦发表，再让撰稿人做任何事情都为时已晚。

但是通过在线回应你还能做一些补救。首先，如果这篇文章是在线上发表的，你可以找到这篇文章并通过它们现有的评论系统发表评论。评论务必清晰、实事求是，同时你应在评论中表明身份，这样会显得你坦率，别人也更愿意接受你。过去我还会在这些论坛中直接公布我的邮箱以回应一些有争议的问题，它收到的成效很少让我失望。

其次，你还可以去回应其他用户发表的评论，尤其当你有一个用户群组的时候。你在产品发布后最值得马上去做的事情之一，就是处理用户的请求，与用户在在线群组中交流。用户会告诉你哪里做错了，哪里不能用（尽管你认为这是可用的），哪里还需要加点东西。如果在产品发布后的几周时间里去密切倾听用户的声音，你便能够基于真实用户的反馈来调整产品路线图，而这将使产品的第二版大获成功。

7.8.5 庆祝发布

每一个令人瞩目的产品发布都离不开团队成员做出的点滴牺牲（有些时候还是巨大的牺牲），因此感谢你的团队为之付出的心血是非常重要的。不要吝惜任何赞美之词，它会让你的团队欢欣鼓舞。举个例子，在为谷歌的“适用于 Microsoft Outlook 的 Google Apps 同步插件”工作 2 年后，我把平时分享给团队的一些早前定下的基础指标的数据合在一起。我可以借助这些增长的指标来表扬团队这两年的出色成绩并增加他们在高管面前的曝光，反过来团队成员可以将这些数字写进他们的晋升案例。如果晋升成功，他们会更加喜欢与你一起工作。

在亚马逊我会为每次发布准备一瓶香槟，然后团队每个人在香槟上签上自己的名字。所有签了名的酒瓶都被当做纪念品陈列在办公室的架子上，用以提醒我们当年的辉煌成就。

理想情况下你需要在发布之后立即举行发布庆祝会。这时候进行庆祝有助于人们将奖励和感谢与牺牲和成就联系起来。但是请注意，不要在发布的过程中举办庆祝会。这似乎有点不合情理，但我曾经在亚马逊见过一个项目经理在软件推出后就举办了发布庆祝派对，派对上有食物、啤酒、纸盘（在亚马逊这算相当高端的派对了）。我问他：“进行得怎么样了？”他以为我指的是派对，便马上回答：“好极了！”其实我问的是发布。

我沿着走廊向前走，发现他工程团队的大多数人都坐在工位上观察着服务器的运行状态。想想这批人会多么泄气，居然参加不了他们自己的庆祝会。还有另一种观点认为不宜过早进行庆祝。你经常需要将发布回滚，或者再冲刺数天以修复新出现的产品问题。因此你最好等你的团队度过手忙脚乱的局面后，再庆祝发布以及感谢他们做出的努力。

嘉奖先进个人也很重要，但公开嘉奖具体的某些人既有好处也有风险。这样做之前你需要想清楚有什么好的理由支持你公开嘉奖。我有时候这样做是因为我认为任何事情都是教学的机会，公开嘉奖这些杰出个人，可以让团队明白做到什么程度才算杰出，这样他们就有了明确的前进目标。

不过公开嘉奖一定要谨慎。曾经有个谷歌副总裁解释他为什么给他的团队一个超级大方的红包时称：“他们工作非常努力，我想做些事情回报他们。”这句话显而易见会被全体谷歌人理解为“我的团队比你们工作更努力哦”！真是躺着也中枪。他如果将这个红包与一个特别的成就联系起来会好得多。当然，如果一个人在某些事情上表现特别突出，公开嘉奖他也没有任何风险，那么就公开嘉奖吧。但如果还有些不妥，不妨私下邀请他共进午餐，然后当面感谢他所做的努力吧。不要因为你的考虑不周而令团队成员蒙羞。

第二部分

掌握卓越技能，更胜一筹

SHIPPING
GREATNESS

你可能在交付软件方面已经做得相当不错了，但我打赌你能做得更好。比如你可以始终更有效率一些，在沟通上也许可以更清晰一点，工作压力可以调整得更好一些，手下的工程团队可以规模更大一些，对高层的影响可以更强一些，对系统设计的理解可以更深一些。如果这些都做得足够好了，你可以略过整个第二部分并把简历发给我。

大多数人仍然需要在交付软件的过程中努力学习一种或多种上述的技能。交付软件不同于其他大多数工作，它需要精准的技术沟通，跨多个领域的深厚学识以及无畏的勇气。本书在第二部分各章节强调的技能都是经过实践检验的、能够提升效能和幸福感的方法，而更高的效能和幸福感又能反过来推动交付。

因为你无法单枪匹马地完成交付，第 8 章会教你如何雇佣或组建一支团队，当收购可行时如何进行收购，当你无法在本地组建团队时如何与远程团队一起工作，以及当已有合适的团队时如何加入他们。

第 9 章讲述了一些你有必要了解的技术知识，这样你才能与第 8 章中提及的受雇工程师们进行有效协作。这一章不是计算机科学教程，更确切地说，它是关于你需要了解的各种面向系统的知识的快速概览，了解这些知识可以帮助你引导工程团队作出正确的决定。也许你会因为拥有卡内基梅隆大学计算机科学的硕士学位而认为这章可以略过，但请切记学历不代表能力，我见过像这样非常聪明的计算机科学硕士毕业生一样会创建一些很不靠谱的架构，所以本书增加了这个章节。

因为无法教你如何想出一些你的团队能够实现的卓越想法，所以第 10 章会侧重阐述有了卓越的想法之后你该如何尽量有效地传达给别人。这章的内容包括如何撰写完美的邮件，如何通过出色的演示来达到目的，以及如何有成效地召开 5 类会议。

当然，在你传达你的产品和进展给别人的同时，别人也会给你一些反馈。有些反馈很有价值，有些则十分荒谬。第 11 章会教你一些应对不靠谱的建议的方法，并探讨你该采纳什么建议以及团队该如何做出正确的决策。

一旦学会了如何组建优秀的团队，理解了技术并具备了优秀的表达能力，你就离成功更近一步了。然而你每天仍需应对一些交付带来的特有挑战，如特性需求和高层的指手画脚。优雅地应对这些挑战能降低你的皮质醇水平并延长你的生命。第 12 章

会探讨如何在日复一日的交付过程中做到游刃有余。

如果一路坚持下来，你会意识到交付是多么地有用，并会做好准备再来一次。第 13 章讲述了该如何再进行一次交付。

如果这些技能都很好地掌握了，你会变成这个宇宙中最强大的团队主管之一。我完全相信这一点。掌握这些技能最大的挑战不在于理解它们，而在于始终如一地在实践中应用它们。祝你好运！

第 8 章

胜在团队

8

SHIPPING GREATNESS

在制定了卓越的产品方案并得到了强有力的组织保障后，你需要组建起一个队伍来齐心协力地构建并交付该产品。这是在你选定想要解决的恰当的用户问题后，所需做的最重要的事情。一个杰出的团队最终将成为你的长期竞争优势。最重要的是，一个杰出的团队能在接下来的软件开发的各个环节中为你排忧解难。

想想看：如果你有一个糟糕的设计师，你会被用户的反馈搞得头晕脑涨，最终不得不多次重新开发同样的功能。一个二流开发主管搞出来的劣质系统设计会导致各种匪夷所思的崩溃，不单用户会产生负面看法，开发工程师们也会被迫加班加点，从而互相辱骂，甚至一听到手机响就烦。一个差劲的产品经理产生的影响最为恶劣，他会不停地用一堆贫瘠的、绞尽脑汁想出来却依旧糟糕透顶的想法来扰乱你的团队。

而拥有一个杰出的团队是令人愉快的，它能创造收益，会让你觉得任何事情都不在话下，你们之间还可能会产生终生的友谊。你必须拥有这样的团队，而且无论头衔是什么，你都可以在团队组建过程中扮演主要角色。下面是具体的做法。

8.1　如何组建一支团队

为了组建一支高效的团队，你必须找到能默契配合的工程主管、产品主管和设计主管。当发现这些人时，你要巴结好他们，哪怕给他们写赞美诗、买糖果甚至洗车都行。你的效率源自于团队的运作，找到一个能带好他们的主管将从根本上减轻你的工作量，还会极大促进你在其他方面努力的成效。因此，纵观整个行业，你会发现不管是在业务中还是在项目中，优秀的人总是聚在一起工作。

你不可能花费大量的心思来关心团队的素质以及团队成员的幸福感。推荐你阅读贝弗利·凯和沙伦·乔丹－埃文斯合著的《爱他们，还是失去他们？》并实践书中倡导的方法，即使你不是一个管理者。这本书谈论了如何留住最好的员工并让他们发挥出最大的能力，并提供了一些好的想法和技巧，以帮助你采取行动以及了解哪种行动可以收到成效。团队主管可以对成员的职业生涯和幸福感施加可观的影响，如指明正确的项目方向、识别问题以及给予渴望获得认可的成员以赞誉。

通常团队主管的工作量很大，单靠一个人很难完成，也很难做好。你需要雇佣一

名工程经理、一名产品经理、一名项目集经理、一名项目经理或者集四种职能为一体的人。为简洁起见，本书中把这类人叫做 PM。

雇佣 PM 至关重要。雇佣一个错误的人来充当团队的 PM 会让你痛不欲生，就像有一把撒了盐的刀每天捅一次工程团队的集体肾脏。这个错误的人选会将产品带向糟糕的方向，会欺上瞒下，会拖延任务，或者虽然完成得很快但质量奇差，还会引发工程团队内部的极大焦虑。不消说，你一定要谨慎地选择，宁可不雇，也不能雇佣错误的 PM。

如果你需要雇佣一个 PM，首先需要明确你自己将承担什么角色。通常在软件业中有项目集经理、产品经理、项目经理和工程经理等四种角色。每种角色的实际定义取决于每家公司的设定，但区别它们还是有一些普遍的指导准则的，下面将一一说明。

8.1.1 项目集经理

我曾经面试过微软的一个项目集经理职位，结果被拒了。面试官当时问我：“PM 该干什么？”这更像一个关于这个工作的难点是什么的面试问题。我怎么回答的呢？“PM 负责交付软件。”

更具体一点，项目集经理的职责重点在于整合不同团队和不同工作职能。步行指数网站 CEO、前 Madrona 投资集团投资合伙人乔什・赫斯特曾经说：“项目集经理担当黏合和润滑的角色，他将不同职能的人黏合到一起让组织能够正常工作，然后再加以润滑以使组织运行得更好。”

另一种看待项目集经理的方式是把它看做是一个比产品经理更少关注业务、比项目经理更少关注项目的技术角色。因此项目集经理产生的交付物比其他一些角色更为模糊，但明确的一点是他会专注于“黏合和润滑”。

8.1.2 产品经理

通常产品经理的职责更偏重软件的业务方面。甚至有些产品经理不负责软件，他们是典型的 MBA 出身，专注于品牌管理、定价、市场进入策略等。你的软件的产品经理将负责所有这些工作，还需要努力从用户的角度出发来帮助确定特性开发的优先级。

在谷歌，产品经理需要做除了写代码之外的任何事。即便如此，我依然知道不止一位产品经理（包括我自己）连代码也写过。在 Facebook 和新兴的硅谷创业公司，产品经理的职责倾向于与谷歌一致，大概是因为这些创业公司和 Facebook 中存在大量的前谷歌雇员吧。

8.1.3 项目经理

项目经理的主要职责在于排定项目计划和协调团队工作，在谷歌这个职位也被称作技术项目经理。他们负责向工程师要评估值，辨识从属关系以及弄清楚如何在更短的时间内做更多事。想知道卓越的项目管理能力能带来多大价值吗？在 2007 年，一辆携带 8600 加仑汽油的卡车在加利福尼亚 80 号州际公路与 580 号州际公路交界的斜坡上翻车了，破坏了一条主要干线，导致当地交通陷入极度混乱之中（http://en.wikipedia.org/wiki/MacArthur_Maze）。初步估算要修复损害需要超过 1 千万美元，但是 C. C. Myers 有限公司大胆地报出低于 90 万美元的价格，以及一条每提前一天完成可多获得 20 万美元额外付费的附加条款。由于加利福尼亚州政府预估该损害每天会造成 600 万美元的经济损失，因此这个报价看起来十分合理。最终 C. C. Myers 的全体人员通过极其出色的项目管理和勤劳踏实的工作，提前整整一个月完成修复，赚到了 500%（500 万美元）的额外收益。

8.1.4 工程经理

工程经理常常是由老牌的程序员担任的。最佳的工程经理是那些由于热爱团队、善解人意、精通交付并乐于构建卓越产品而晋升到该职位的人。最差的则是那些只想获得更多权力和薪水的不得志的工程师。你知道自己喜欢的是什么，让另外那种人出局吧。

产品经理、项目集经理或者项目经理，甚至是技术项目经理都可以是工程经理的属下，但也可以是合作伙伴。一些工程经理认为他们的主要任务是维持工程团队的幸福感，另一些人则认为是通过雇人、走流程或者其他方法来保持工程质量。还有一些人认为他们的角色近似于产品经理，但拥有工程资源的使用权，这样他们可以去构建任何想要的东西。

每个工程团队都需要工程经理，但不是每个团队都需要产品经理、项目集经理或

者项目经理。因此，如果你是其中的一员，就必须配合好工程经理，如果你的工程团队训练不足或缺少质量管理流程，或是你的团队成员不开心，你将难以完成交付。同样的，如果你是工程经理，找到能弥补你的弱项并让你更快更好交付的人相当重要。

8.1.5 如何雇佣产品经理、项目集经理或者工程经理

雇佣团队主管很难。我相信对于谷歌、亚马逊、微软以及其他类似的公司来说，仅靠面试来寻找优秀的主管是十分可怕的，这从他们如何评估候选人就可以看出来。例如在谷歌，影响雇佣的最重要因素是候选人的平均绩点、毕业院校和内部推荐。这些因素都不是面试过程的一部分，这应该可以给你一些启发。在亚马逊，候选者的背景必须十分专业，但他们面试被问的问题只需要一点点知识就足以回答了（更多内容见第 9 章）。我认为这种雇佣主管的过程在很大程度上是不靠谱的，但它是团队主管工作的主要部分，你需要知道该怎么去做。下面有 5 个主要的雇佣主管的原则。

- 雇佣比你聪明的人。
- 雇佣懂得自己不是来当老板的人。
- 雇佣表达清晰、言之有物的人。
- 雇佣用数据说话的人。
- 雇佣充满活力的人。

1. 雇佣比你聪明的人

首先，你应该雇佣比你聪明的人。然而谷歌前综合项目集经理迈克·史密斯指出："这句话忽视了人类的基本本能……（你希望）施加控制"，他曾负责协调谷歌西雅图 / 柯克兰办公室的产品经理雇佣工作。因此，如果你乐于放权并相信他们能够胜任队伍领导者的工作，你就必须且只能雇佣比你聪明的人。我坚信埃里克·施密特也信奉并身体力行该原则。我曾听他说："谢谢你无视我，这就是我雇佣你的原因。"但郁闷的是，他这句话是对别人说的，我可没胆子无视埃里克，他比我要聪明得太多太多了。如果你能雇佣比你聪明的人，并授权他们可以无视你，那么请继续往下看，否则你还是不要雇佣任何人，自己一个人做得了。

一旦决定要雇佣比自己聪明的队伍领导者，你就会想要了解如何评估一个人是否聪明。我极度不喜欢微软的问"脑筋急转弯"一类问题的评估方式，我认为它们实际上测不了人的聪明程度。例如下面是一个我在微软面试时被问到的真实问题。

问：你在一艘船上，举着一块大石头。如果你把石头扔到湖里，湖的水平面会发生什么变化？

答：它会下降，因为石头在水中只占据了自身体积大小的空间，而它在船上时会挤占同等质量的水的体积大小的空间。

问：<长长的停顿>等下，你之前听过这个题目吗？

答：没有，但是我学过工程……

这类方式唯一有趣之处在于观察候选者在被问住时会有什么反应。若能在没有帮助的情况下破解难题，那就意味着他有一个聪明的大脑。除此之外，人们这种解答脑筋急转弯的能力与他们寻找问题根本原因、准确评估行业或者思考产品市场竞争力的能力基本无关。下次请不要再问这类问题了。

有些人会质疑我上面的回答："是的，但假如那块石头是浮石，它实际上能浮起来呢？"我会对他们说，这有意思吗？

最佳的检验纯粹智慧的方式是背景调查。《执行：如何完成任务的学问》一书的合著者拉姆·查兰会坚持亲自做候选人的背景调查，我认识的一些优秀的 PM 经理也会这么做。如果从流程上看背景调查还为时尚早，那么改为核查候选人的简历也是可以的。在名牌学校拥有很高的平均绩点是额外加分项之一，它预示着候选人能够胜任该角色。

实际推出过很多产品是关键的加分项，说到底，如果想要交付出卓越的产品，有过交付经验还是非常重要的。一个主管候选人的简历如果又长又空洞，那么他出局的结果就基本注定了，但如果他仅提供了一份单页的简历，但在简历中重点突出了他推出过的产品及其对财务或用户体验的影响，那么他是一个"必须面试"的人。

好吧，你也许还未被说服，还是准备提出经典的大理石称重问题（有 24 块大理石，其中 23 块大理石的重量都是一样的，剩下一块大理石偏重。请只使用一座天平找到它）或它的一些变种。让我帮你排忧解难，潜在的候选人听好了：这个问题和所有这类问题都与二分查找有关，最快找到的次数是$\log_N^K$（K 为大理石数量），当大理石分成 2 堆时 N 等于 2，分成 3 堆时 N 等于 3。

现在你可以不要再问那些愚蠢的问题了，安心去询问候选者的平均绩点、做背景调查吧。

2. 寻找那些懂得自己不是来当老板的候选人

你应该雇佣那些懂得自己不是来当老板的候选人，即便你要雇佣的是一名工程经理。你希望你的工程团队能够做那些他们认为是对的事情，并能从工程经理那里获得支持，而不是命令。判断 PM 是否懂得这一点的唯一方法是与他当面接触。

我在面对面的面试时会抛出一个从麦肯锡公司那里借来的问题："讲一个你是如何改变某些人想法的经历吧，以及你使用的技巧。"我等着听候选人是如何应对这种不在掌控的情况的。我会注意其中是否有合作决策（见第 11 章），是否有基于数据的争论（见第 6 章）和聪明的向上反馈（见第 11 章）。如果候选人说"哦，我只是劝技术主管去试一试"，那么这很可能意味着这个主管曾经把自己看做是老板。

3. 寻找那些表达清晰、言之有物的人

下面举一个例子，说明当你雇佣了一个沟通能力很差的人时会发生什么。

我：<升级>完成了吗？

@amazon.com：嗯。

我：比如说是测试完了并开始运行了？

@amazon.com：啊，没有，不过它看起来很像 Java 了。

以上是真人真事！我需要主管们给我实际具体的回答，因此当我紧接着问关于详情的问题，如"你是如何说服拉里批准你的发布的"。差劲的回答是："我们讨论了这个事情，然后他有点儿转变了想法。"出色的回答是：

首先，我把所有相关人召集起来开了一次会议，会上我们达成了共识。然后我请求我的高级副总裁帮忙给拉里吹吹风。等到拉里被说动后，我给他做了 10 页的演示并倾听了他的反馈。在会上我们解除了他的担忧，因此他批准了我们的发布。

请注意，后面这个回答既简短又具体。它有头有尾地讲述了他做了些什么、他是如何与团队协作的。

4. 雇佣习惯用数据说话的候选人

一个好的技术主管应该具备独立计算的能力。我会给那些使用数字的候选人打上不错的分数，即使他只是在提及演示的页数时使用了数字。我偶尔会问候选人关于市场规模的问题，但目的不是评估他们能否细分市场，而是看看他们是否善于估量和独立计算。一个典型的关于市场规模的问题可能是"在美国一款新的智能手机的市场有多大"。大部分工商管理学硕士在梦中都能回答这类问题，因为他们在出来咨询如何

面试之前，就接受过关于这些事的训练。下面是我对这个问题的回答（虽然不能保证答案一定完全正确，但是你会看到它是如何证明我对数字毫无畏惧的）。

美国有 3 亿 5 千万人口。我估计这 3 亿 5 千万人口中 12 到 75 岁的人会对手机有需求。因此在美国手机的总体市场约为 3 亿用户。

如今智能手机的用户是有差别的，可以细分为电话用户、社交媒体用户和商业用户。

社交媒体的用户集中在 12 ～ 30 年龄段，约占市场的 30%，即 9 千万。再加上商业用户占 30 ～ 60 年龄段的 50%，于是我们拥有了另外 30% 市场的 50%。

9 千万的一半是 4 千 5 百万，加上它之后，智能手机市场总共有 1 亿 3 千 5 百万用户。3 亿用户中剩下的部分可以归入电话用户，这部分人不用算在内。

因此有 1 亿 3 千 5 百万人可能需要一台智能手机。我认为 Apple 正在计划销售 2 千万台 iPhone 5s，而 Android 是它的两倍，因此这个估计大体上是准确的。

如果我们要推销一款新手机，销售对象很可能是这两大群体：新智能手机用户和升级用户。因此，如果假定已经有约 4 千万台 iPhone 和 8 千万台 Android 机被销售出去，那么等于有 1 亿 2 千万人已经拥有了智能手机，只剩下 1 千 5 百万人可以成为新用户了。而在 1 亿 2 千万老用户中，保守估计，假设每 3 年换一次新手机，那么 1 亿 2 千万用户中的三分之一将在一年内换手机，那就是 4 千万。

换句话说，我认为在第一年会有 5 千 5 百万台手机的潜在市场规模——4 千万升级用户和 1 千 5 百万新用户。现在，能卖多少台手机就变成了一个很不一样的故事了！最后可以说一下手机这个业务有没有吸引力呢，它有多快的增速呢。

这就是回答市场规模相关问题的方式。先假设一个数值，接下来用其他数据来检验你的假设。使用约略的整数，通过理性的市场细分来削减预估值，然后得到一个真实的数字。最后，表现你对这块业务的热爱与激情，即使你可能不想在一家问这种问题的公司工作。

5. 雇佣充满活力的人

主管常常是团队背后的驱动力，如果他们不能给团队带来活力，你的目标就会落空。如果你提出使命的目的之一是唤起人们的兴趣（见第 1 章），那么传达使命的最佳人选是那些能鼓舞人心的人，而这种能力来自于活力。要在面试中判断候选人是否充满活力，一个可能观察到的迹象是他是否有奇思妙想，因为乐于投入大量精力来思考你提出的具体问题的候选人，也更可能投入大量精力到他自己的团队。我

还会看在围绕问题解决过程中他是否有兴奋感。我提出的设计问题都是我很感兴趣的，优秀的候选人能在和我讨论问题的过程中兴奋起来，并和我一起探索这个问题。

8.2 如何收购一家公司

一家大公司常常会在项目初始阶段收购一家较小公司，这样就拥有了一个已经武装好的团队。一个软件项目主管也常常会为了解决问题或者更快进入市场而寻找可能的收购机会。但收购通常都不简单，懂得如何恰如其分地运用它很重要。

通常考虑收购一家公司会出于四个目的。

知识产权

你能使用这家公司构建的技术、内容和专利。

人才

你能使用这家公司雇佣的人才。

客户

你能凭借这家公司的客户来加快业务增长。

防御

你买这家公司是为了让别人没法买它。

关于这四个目的，在微软和迪士尼做过收购工作的迪士尼工程副总裁迈克·史密斯说："你可以期望能达到其中一个目的，如果运气好，两个也有可能，但如果想通过收购达到两个以上的目的，那你十有八九会大失所望。"

8.2.1 知识产权收购

在正式开始考虑收购技术或内容之前，你需要简单计算下是自己构建的成本低还是买别人的成本低。算法很简单：多少个工程师需要花多少个月才能开发、测试并交付类似的软件？将所需工程月数乘以一个工程师满负荷工作一个月所需的成本，再减去因整合被收购公司的知识产权而产生的成本——这个成本也可以用工程人月为单位来衡量。最后算出的结果就是你可以接受的收购金额，这里假定进入市场的时间并不重要。

但是，进入市场的时间通常都是很重要的，因此你需要粗略计算下收购并完成整合所需的时间能比自己开发少多少，潜在销量的价值有多大。你也可以直接取 6 个月的销量，这差不多等于你能取得的收益。将这个数字加入到你的第一次评估中去，你就能知道交易的全部价值了。

如果你觉得可以低于刚才计算的金额达成交易，那么继续推动收购就是明智之选。

接下来需要仔细检查即将收购的软件。你不能相信这家公司的的代码质量，就算能相信他们的代码质量，但还是需要验证。你需要找一个未来不会涉足这个业务的高级工程师去检查代码。创业公司会因为他们的秘密武器被揭开而焦虑不安，你的律师也会担心万一收购不成会给团队带来污点。但这些担心对你没有任何帮助。你必须让自己和团队信任的人去检查代码和架构。如果不这样做，你就等于在机械师都未检查过的情况下买了一辆租赁车。

如果你检查完代码后觉得它的质量还是可以的，接下去就需要拟定一个计划来确保团队和技术的整合。整合项目与大部分项目一样，所花的时间会超出你的预期，甚至还会远远地超出你的预期，因为你需要与新的成员、外部的服务器、不同的软件授权以及各种没有文档的细节打交道。

8.2.2　人才收购

人才收购是所有收购中最复杂的，这并不奇怪，人是复杂的生物，而人才收购全部与人相关。你必须像平时那样通过面试来评估打算收购的人才。你面试的人越多，得到的信息就越充分，收购的风险就越小。但与之相对的是，你面试的人越多，对双边业务造成的干扰也就越大。所以你必须谨慎地进行面试。

这里有一点需要告诫你：不要在人们不知情的情况下进行面试。这样会产生适得其反的效果，我知道曾经发生过至少一例这样的事情。负责交易的团队最终不得不重新面试一轮，这为交易笼上了阴云。

面试还会帮助你了解雇员适合的岗位。我将这些人才分成三大类。

关键人才

他们是那些闪闪发光的人，缺了他们，你就得马上找人替代。但他们的专业功底深厚，你都很难找到类似的人来填补他们遗留的空缺。

好的雇员

他们是一群你乐于招进来做当前业务的人。他们是一流的候选人，但这种人你自己花个把月也能找得到。

多余人才

他们是一群达不到雇佣要求的员工，你需要二选一：和他们续约一段时间以使你能平稳将他们开掉，或者立即终止与他们的雇佣关系。这看起来是个艰难的过渡，但收购的现实就是这样。

流程、面试和交易估值的主观性使得人才的批量收购非常艰难，即使收购成功，人才的整合也会非常不容易。迈克・史密斯分享了一个他在微软经历的悲惨往事。

我曾经推动收购了一家名为 Conversagent 的公司（成为 Microsoft Windows Live 的代理商）。在人才评估时，我们清晰地意识到再雇佣 20 ～ 30 名工程师才能满足业务运作的需要。收购在附加该约束条件的情况下通过了，但随后对方财务却决定拖延扩充人员编制以抵制再雇佣新人，整整拖延了 9 个月。核心人才在第 12 个月离开了，没有人员填补。收购最终失败了，它目前的价值还不足当初收购价格的十分之一。

这个故事告诉你获得所有合作方的支持是多么重要，你需要在交易结束之前就让他们（包括 HR、法务、财务）参与到你的人才收购中来。

8.2.3　客户群收购

如果你能从每个用户身上赚到 2 美元，而且正要收购一家拥有 1 千万用户的公司，你应该能借此获得 2 千万美元的收入，对吗？错，你只能获得其中很小一部分。这一部分的大小取决于你借助这家公司的业务做什么。

如果打算收购的业务是自我维持的，你可以让它保持现有规模运作下去，并尝试对该业务的客户追加销售你的新产品。然而转化率可能会很低。如果这 1 千万用户有约 20% 的转化率，那么这笔交易就值不到 4 百万美元。

如果打算关闭这个业务并将它的用户转移到你这儿来，你可能要付出相当大的成本来关掉它，且在这个过程中还会流失客户。预计按照这种方式完成的交易最后的价值只有其潜在价值的 50% 左右，即大约 5 百万美元。

这些收益都太低了，所以更大的可能是，你将收购客户的交易看做加速销售的手段，以应对市场高度竞争因而快速扩张成为当务之急的局面。如果身处这样的局面，

你也许希望赶紧跳出苦海以免被压垮。或者至少吃点泰诺，这样你的胃会感谢你的。假如不打算跳出苦海，你就需要通过预估销售增长来对这笔交易估值，这种算法充满了不确定性。

不管你对这笔交易如何估值，确保你正在使用的是正确的数据基线。这意味着你需要查看日志。除了查看印象数和注册用户数，你还要查看“7 天活跃”用户数。你还想要回头客，因此你需要测量回访客户数和参与度（网站停留时间或者 App 使用时间）。让你的团队检查日志数据，或者至少你要了解生成报表的系统，不管这个系统是 Webtrends 还是 Google 分析。

8.2.4　防御性收购

我从未主导过防御性收购。在我看来防御性收购不见得很好，这种决策带有一股恐惧的味道。如果你有钱进行防御性交易，请不要故意作恶。

如果正在考虑进行一次防御性交易，你可能需要想一想什么行为会涉嫌垄断。你最好习惯在说话前申明“我不是个律师”，这样至少你发表的意见有一层保护。我不是个律师，所以无法告诉你什么是不能做的。

8.2.5　收购的陷阱和最佳实践

最后这里还有一些建议和警告，你在考虑收购一家公司时需铭记在心。

1. 计划将你团队的部分人员调入他们团队

将你的一些高级职员调入收购的团队十分重要，而且会发挥很大的作用，因为这样会把你的商业文化、商业实践和商业政策带进这支新团队。除此之外，一个优秀的精力旺盛的开发主管还能破开坚冰，并帮助这支团队更快地提高效率。效率提高了，人的幸福感就会增加，结果这个新团队就会成为一个愉快的团队。你应该按照大约 10:1 的比例来为收购而来的工程师配备高级工程师。如果无法匀出任何人，你就需要想办法从新团队调人到你现有团队做开发主管，然后把原先的主管调到新团队去。

不要低估文化适应的重要性，也不要低估新人适应领导风格所需要的时间。如果你不管不顾，就会催生出一个难以驾驭的工程师团体，他们一心打算熬完一年后将股份变现走人。这些人工作在痛苦中，还会对其他业务产生恶劣影响。所以你要确保新

老团队紧密融合。将最优秀的工程师调到新的团队会对此产生巨大的助力。

2. 计划整合产品

你不仅需要知道如何将他们的服务器架设在于你的虚拟 IP 之下，还需要知道如何处理他们的品牌以及如何将双方的计费系统打通。很多收购原本可以更为成功，可惜在业务整合上收购者们付出了太多的工程成本。清晰的计划还有助于团队的过渡。

3. 了解之前所有的交易和负债

在交易晚期才发现创始人欠了别人一百万美元可不是件令人愉快的事。这不是你的问题，但出于某种原因，创始人总是希望你能帮他解决这类问题，所以这也变成了你的问题。在确定收购金额之前，通过会谈了解清楚该公司的欠款、负债以及任何签署过的交易是非常重要的。一个好的律师会帮助你做这些事，但你最好也做一下这方面的功课。

8.3 如何与远程团队合作

即使在天时地利人和的情况下和一支优秀的工程团队合作也是件非常不容易的事。如果这个团队还不在一处办公，合作就更加困难了。要是互相之间还有 12 小时的时差，那合作更是天方夜谭了。跨多个时区（如悉尼、斯德哥尔摩、印度和美国西海岸）协作的情况从根本上来说是滑稽可笑的。我曾经领导过这类分散式的项目，至今回忆起来还觉得可笑。

远程（谷歌内部叫做“分散式”）团队是当前软件开发中的必要之恶。人们愿意使用远程团队的原因之一在于各地区都有着自己的特色，这样能聚集更多志同道合的精英。例如，特拉维夫市和斯德哥尔摩盛产视频专家，罗马尼亚盛产安全精英，大规模分布系统的天才多是来自像匹兹堡和西雅图这种地方的大学城。谷歌超过 50% 的雇员都在山景城总部之外工作。Facebook、谷歌、Ticketmaster 以及日益增多的旧金山湾区公司都在西雅图开设了分部，而这只是其中一个例子。美国的移民政策虽然使得一些世界顶尖的工程师人才很难来美国工作，但这并不妨碍他们为美国工作。

根据这些行业趋势以及你对优秀工程人才的无尽渴望，你终会有一天需要考虑与家乡之外的工程师合作。这很有挑战性，但你可以做一些事情来使生活更轻松一点。事实上有 9 件事情：

- 组建一支工程师团队
- 充分沟通
- 不要外包设计或 PM 角色
- 适应文化差异
- 构建清晰的需求
- 忍受时差，通过任何方式会面
- 委任得力的主管。
- 大量出差，或者完全不出差
- 与远程团队共饮

8.3.1　不要租用工程师——组建一支工程师团队

工程项目是长期而复杂的，个体之间的协作对它意义巨大。利用由协作产生的动力的最佳方法是组建一支包含至少 3 名工程师的拥有共同纲领的团队。我把 3 名工程师的数量称作“临界数量”，意思是说这个数量的团队刚好可以产生内在的动力。共同纲领能给予团队方向感，它还帮助团队独立进行决策，当缺乏时间照管他们的时候你会需要它的。例如，当一个开发者完成了一个项目或者卡住了的时候，共同纲领会告诉他下一步该干嘛。为团队定义清晰的共同纲领还能帮助他们减少对未来的焦虑。

8.3.2　充分沟通

我注意到在与远程团队合作时有一个常识性的真理：离你越远的团队越焦虑。例如你在加利福尼亚办公，纽约的团队会想当然地认为有些事情没有沟通好，然后搭飞机来加利福利亚跟你大声抱怨。甚至悉尼和印度最优秀的工程团队也会真的慌张起来，因为他们离美国是如此之远，以至于想当然地认为他们被误解、不受重视以及被排除在决策圈之外。为了减轻这种感觉，你能做的最有效的事情是充分地、反复地沟通。

使用 Skype、Google+ Hangouts、WebEx 以及你手边能用的任何工具来提升与远程团队的沟通质量。因为开发者讨厌打电话，所以对你来说减少初始的摩擦真的很重要。我在谷歌曾带过一个成员分散在西雅图和山景城两地的团队。我们为每个团队购买了专门用于视频会议的小型设备，这样可以快速将其他团队拉入每日例会或者随机的设计讨

论中。这真的非常有用。你也可以通过 Google+ 的 Hangouts With Extras[①] 做到这一点。

8.3.3　尽量不要外包设计和 PM 角色

外包设计师也可能把工作做得很出色，但你显然得不到其他巨大的价值。一个优秀的设计者会解决众多你想都没想到的问题，当他全面了解了你在做的事情后将表现得更为完美。你能在第 3 章找到更多关于如何与设计师合作的信息。请尽量从内部雇佣某个人来承担设计工作，要外包也只外包视觉设计。

同设计师一样，若是产品、项目集以及项目经理能够全心投入到团队中去，他们将发挥出极大的价值。例如，他们会从无心听到的只言片语中发现传达上的错误。他们会发现工程团队被卡在哪里了。他们会反复重申你的使命和策略，这样团队不会偏离方向。他们会建立与工程团队的个人关系，这样工程师在评估任务量时会感到放松。出于这些以及更多的理由，我无法想象将产品、项目集或项目经理外包后会产生什么后果。

8.3.4　重视文化差异

我曾经历过一轮发人深省的个人绩效评审。我与一名工作非常出色的女性工程师紧密合作过，为保护隐私起见暂且称呼她为莎拉。莎拉有着强大的领导力，写出高质量的代码，而且我觉得她很有想法，我们合作让产品取得了不错的进展。我还是她晋升的有力支持者。你肯定也能猜到她会如何评价我，对吧？

莎拉的工程总监和我坐下来聊了一天，他告诉我说我有一个问题，莎拉觉得我不注意倾听她的想法。我当时就震惊了。这位非常聪明的华裔工程总监给我解释了一番，他说："你应该考虑到两个情况，莎拉是个女人，她还是个华裔女人。她经历过很多不被倾听、无法坦率直言的往事。而你是个说话真的非常大声的身高 1 米 93 的白人。"

文化差异非常有趣。我当然不会因为莎拉的种族或性别而区别对待或带有偏见，这是愚蠢的、罪恶的且非法的。但在类似的情况下，当与一个比起英语还是更懂 C++ 的罗马尼亚大个子白人一起工作时，我会尽量避免使用任何常青藤派头的单词，这只是解决实际问题的智慧。所以适当调整与莎拉这种独特听众的沟通方式是言之有理的。

这只是文化差异的一个例子，我还能列举很多。例如，我发现相较于美国的团队

---

①Hangouts 加强版，包含一些实验性功能。——译者注

来说，苏黎世的团队对得出“正确”的解决方案的关心程度远远超过构建出原型。我在西雅图的本地团队起初很是恼火，但当我们理解了这是苏黎世团队的做事方式后，在接下来的很长一段时间我们都配合得很好，这种理解帮助我们给了瑞士团队他们所需要的空间。

你不需要理解每种文化的独特之处，但需要认识到美国东海岸团队的规矩和西海岸是不同的，西海岸和英国也是不同的。你必须积极了解那些可能有差异的地方，然后想办法去平衡。你能通过在大量的沟通中寻找反应模式来着手了解这些差异。

8.3.5 构建清晰的需求

新团队不管位于何处，都会有些相似点。其中一个相似点是他们真的不知道该干些什么或者为什么该干这个。而且对于远程团队这个问题会更大，因为你无法坐在他们的办公室中间为他们解答随时产生的问题，每天打电话向你的投资人重复 10 遍你的使命以及反复提醒开发主管你的下一个截止期限多么重要。因为你不在那儿，你需要提供完备的需求说明来充当你的“替身”。如果准备好了第 2 章中描述过的那个产品需求文档，你就会轻松一些。虽然详实的需求对所有团队来说都很关键，但对远程团队更为必要。

8.3.6 忍受时差

12 小时的时差令人厌恶，但这个事实无法改变，你不得不通过某些方式来应对它，因为你需要参加例会以及一些一对一的会议，或者有时候你只是需要花半个小时去倾听某个人谈论他的生活。但这半个小时不属于你，它属于其他人，你也许会感到烦躁并宁愿躲去看《每日秀》。

我只想出了两个办法来应对时差。

- 使用数字录像机。
- 大早上起来工作，或者大晚上工作。无论你更适应哪个，请将你的会议总是安排在一天的开始或结束。

这就是我所做的全部。你只能忍受。花些时间沟通和会面很重要；当时间很紧的时候你很容易忽略远程团队，因此注意一下你的时间安排，匀出一些时间给他们。

8.3.7 委任得力的主管

你也许想立刻出现在世界各地，但最终你只能到达少数几个地方，而且还很可能会迟到。变成一个鼓胀的小球没有任何用处，饮用苏格兰威士忌来释放压力也对你的肝脏不好。你需要委派助理到远程团队所在地，如果可以，从总部委派一个你最得力的主管到新团队呆上 1 个月左右。让得力的技术主管加入到新团队非常有利于文化和流程的移植，这也是我在团队收购中推荐这种做法的原因。

任命一个当地的主管则至少可以让你不用参加一堆深夜的会议，因为当地主管可以和你总部的工程主管一对一地开会交流。这些会议不止能帮助你减少技术会谈的数量，增加文化会谈的时间，还能帮助你减少发现危机所需时间。例如，你将与远程办公室的会议安排在周一，而总部的工程主管将它安排在周三，他将可能比你早整整两个工作日发现危机，因为你安排的会议还要等到下周一。

8.3.8 大量出差，或者完全不出差

如果和经常出差的人交谈，你会发现如果每隔一个星期出趟差，这件事就变得轻松一些。当然出差仍然是件糟糕的事情，但它会更好受一些，因为你可以坐更好的舱位，你的行李始终是打包好的，你的行程状态能让你更快通过安检。你始终住在同一个宾馆，这样你就很少会丢东西。你适应了如何边走边吃，你还艰难地适应了宾馆的健身房。

另一个出差的诀窍是当天来回。我知道当天来回听起来有点夸张，但对于少于 3 小时的旅程来说它的好处很多。你能睡在自己的床上。你不用带换洗的衣服。你能在飞机上处理邮件。最终你会发现这还蛮靠谱的，但要是你能说服自己起个大早，你会发现，比起宾馆的免费咖啡，你会更喜欢这种当天来回的出差方式。

每隔一到两个月远距离出差一次实在令人讨厌。你也许觉得这种低频次会更好，但它在很多方面会更加糟糕。

8.3.9 与远程团队共饮

我有次花了些时间在韩国挑选一家主要的消费电子产品公司以便构建一些特定服务所需的硬件。我和负责业务拓展的同事杰克、他们的业务人员和工程人员一起出去吃饭，在吃饭途中他们向我们讲述了一件有趣的事。

很多韩国人喜欢喝烧酒，它是一种类似于伏特加的酒。我无法适应这种口味，但不管怎样，过长的时差导致我不受控制地喝了一些。那个工程人员接着告诉我们，在韩国，很多工程师都有着强烈的个人意见，但感觉无法在工作场所中表达出来，于是他们会在下班后和同事一起去喝一杯，倾诉心中真实的想法。事实上这也是韩国工程师这么喜欢喝烧酒的原因。

这个故事告诉我们，除了尊重文化差异外，你还应该认识到人们在不同的环境会讲不同的真话。酒吧和饭店就是两个能讲真话的地方（我认为是最好的两个地方），它们都是靠酒精来刺激的。

但这并不意味着你需要成为一个很有酒量的人，意识到这一点很重要。虽然可能有一些与商务场合饮酒相关的国际礼仪，但滴酒不沾未必就不受工程团队的欢迎，他们很可能愿意与你共饮，因为他们真正希望你做的是让他们喝几杯，然后倾听他们的想法。噢，他们还想让你把单也给买了。

8.4　如何加入到一个新团队

无论你是组建了一支新团队，还是从高层空降到火车失事现场来解决问题，都需要做两件重要的事情：弄清楚你应该扮演的角色，以及下好第一步棋。

弄清楚在这个特殊团队中你的理想角色应该是什么很关键。一些工程团队不想要任何项目管理——在这种情况下，你需要弄清楚如何能尽可能少干涉一些，也能跟踪好你的项目。一些工程团队希望你能专注在市场营销上，然而要是你有很深的技术造诣，其他人也会邀请你参加技术讨论并欢迎你发表意见。

为了弄清楚你在团队中应该扮演的角色是什么，你必须敏感地意识到团队需要什么。你可以与团队其他主管一同参加一周一次的一对一会议来为团队把脉，还可以通过与团队每个工程师进行每月一次的一对一纯感情联络会议来建立人际关系。通过透明化所有流程来建立信任也很重要。这些行为会使你专注于能做的事情，并让你成为团队的出色服务者。它们也是一种投资，你将在冲向终点的高难度冲刺过程中获得收益。

在这个节点上你可能会遇到一些小难题。我只遇到过一次，但非常痛苦。你也许弄清楚了团队需要什么，想要什么，但你发现你没有权力去做那些为了出色完成交付而需要去做的事情。这可能是因为你在所有的事情上都必须听从上面的命令，也可

能是因为你尝试过了但阻力巨大，还可能是其他的一些原因。这些原因中一些能被解决，如团队抗拒改变。其他的则难以解决，如你的主管不认可你所坚信的需要做的事情。当碰到这种情况，你可能需要考虑加入别的团队了。

如果你打算呆在团队里，由于你已意识到了需要专注于什么，也收集了一些关于团队及其项目和问题的信息，你将很可能发现他们的产品、项目集或者项目是多么糟糕。大半的时间里，这三者就像僵尸糟蹋过的饭菜。不管这个工程团队有多优秀，只要深究下去就会发现他们大多数项目都是乱糟糟的。

也许因为一些不可思议的原因，你深究下去发现每件事情都是清晰的、令人愉快的，请再检查一遍，然后开始存钱买期权吧。但你很可能像我们大多数人一样发现手头上的事情是乱糟糟的，你需要立刻做两件重要的事情。

第一件事，不要和团队说你们的产品一团糟这种话。他们很可能不这么认为，所以他们可不喜欢你把这个事情抖出来。相信我说的这一点，你不会想要先吃一堑再长一智的。记住，他们做这个的时间比你长，其中会有很多原因使得产品如此糟糕。比如很有可能团队的工程师都十分聪明，只是领导有问题。而领导问题的实质多半又是缺乏领导，所以需要你加入这个团队。当遇到领导问题时，请逐步改善它。

我是个十分直率的人，所以我在职场生涯早期信奉“向权力进言”。这使得我乐于直接指出在一些团队中看到的问题。虽然我尽可能温和地指出来（可惜还不够温和），但依然难以被人接受。我那时是个门外汉，所以我确实不理解“我们做的事情不一样，你需要花更多时间来上手”这种话。有些时候，比如当我对一帮谷歌地图设计师的工作指手画脚时，我还会引发混乱和痛苦。后来我才认识到在大多数情况下你都应该尽量先将事情本身搞清楚，而不是张口就说产品一团糟这种话。

发现问题后你必须做的第二件事情是做一个选择：你是打算延期交付以解决这种混乱状况，还是承认它的存在然后正常交付。做出延期决定的最佳时间点是你刚加入团队的时候。因为你是个新成员，你不对延期负责，你只是指出团队将无法按期发布。如果使用诸如 Bug 数量、工程评估量、假期安排等客观数据作为证据，就能以公正客观的姿态来谈论交付日期。

但有时候你的任务是不惜任何代价完成交付，你必须做到这一点，因为产品的新鲜度和改进关系到用户体验，所以这种要求是有道理的。你应确保没有较大的隐私或安全 Bug 后再推动产品发布，发布成功后再解决团队和流程上的问题。等到这些都解

决后，你再为他们制订一个合理的绩效提升计划，给他们一次机会，若计划结束后依然有人表现差劲，这时候再把他们开掉。

根据我的经验，你将加入的团队可能有五种主要的类型。每种类型都有最佳的应对方式。表 8-1 展示了如何根据他们说的话来识别团队类型并正确回应。

表 8–1 团队类型和回应方式

他们说……	你说……
太棒了！让我们构建出100 个特性来！	放在 V2 怎么样？让我们先专注 V1 中的用户故事。
我们已经搞砸了 100 次了，你这是 101 次。	我知道事情不容易，所以我们要精心制订一个让我们都真正相信的计划，然后把这个计划卖给其他人，一步一个脚印向前推进。
我们知道我们在干什么。你来这里干嘛？	我是来做一些业务工作并帮助你们向上管理。之后我们再一起理一下其他事情（然后说服主管去推动变化）。
嘿，我们老板可没给我们讲过这个事情！	那让我们一起去找老板吧，这样我们信息可以保持一致。
我们干得很快活。我们做了些有趣的演示！	听起来不错。（然后转身回去向高层汇报。对于一个为了赢得世界职业棒球大赛而开心地练习传球的团队来说，挑刺是没有意义的，除非挑刺本身就是既定目的。）
谢天谢地你来了。	谢谢，我很好奇上一个人做了什么对你有巨大帮助的事情，让你对我如此期待？

第 9 章 9

胜在技术

SHIPPING GREATNESS

如果想快速交付一款卓越的产品，你必须会询问富有洞见的问题，会正确地引导方向，并明智地决定哪些事必须现在做，哪些事可以之后再做。你还要会评估和雇佣工程经理。因此，你对技术的了解程度最不济也要与对你车里的汽油的了解程度相当。知道汽油并不意味着你就会开车，但这意味着你知道车最好要加满油，否则你的道奇车就会变成一个超大号砖头。这就是你为了能开车回家所真正需要的全部关于汽油的知识。

虽然你需要足够专业才能做好这些事情，但我相信你并不需要有计算机科学学位才能交付卓越产品。只要你能理解我在这章讲述的系统方法，你就能做到这一点。事实上，我确信要是理解了这些东西，你就能游刃有余地通过谷歌、亚马逊、微软等公司主管级别技术环节的面试。如果你想顺利地通过面试或者从容地掌控产品开发过程，你需要了解四个 S：服务器（server）、服务（service）、速度（speed）和扩容（scaling）。一旦理解了这四个基本元素，你就能够向团队询问恰当的问题了。

9.1　第一个 S：服务器

尽量不要买服务器。首先，这样你不得不学习很多有关服务器的知识，但你学到的所有知识会在六个月内过时，这令人沮丧且效率低下。其次，你不得不做大量的维护工作，如应用服务包、安装更新以及其他琐碎的事情。更令人沮丧的是，在指定和维护自有服务器方面，你的工程团队有数不清的工作要做，这并不是他们最应该做的事。他们最终会得到我刚才所讲的教训：任何学到的关于服务器的知识都会在六个月内过时。因此优秀的工程师会尝试学习如何避免去做运维的琐事。

采用托管主机的办法可以让你省心不少，你也不再需要买瓶威士忌去安慰昨天半夜飞去网络运维中心的臭着脸的开发工程师了。使用亚马逊的 EC2、谷歌的 AppEngine 或者其他类似的服务，你将放弃很多你反正也不需要的控制权，并由此免去很多烦扰。

如果你因为某些特殊的原因而必须拥有自己的系统，那么不妨向提供商租赁它

们。不用担心这个提供商是否是本地的，你需要担心的是当虚拟 IP 出问题的时候，他们是否提供了电话技术支持。对于一个没有拿到计算机科学博士学位的人来说，当虚拟 IP 出问题后，你怎么知道该做些什么呢？所以一个合适的被选中的提供商应该有专门的工程师来处理这种事情。如果没有那就是你的错，你选择了错误的提供商。记住始终做背景调查（更多信息见第 8 章关于如何组建团队）。

通常情况下系统是一个三层架构，如图 9-1 所示。这个架构也许听起来很复杂，但实际上超级简单。



图 9-1 一个简单的三层架构

数据层一般是一个数据库，客户记录等数据都放在里面，你可以使用一些类似 SQL（Structured Query Language，结构化查询语言）语法的语言来检索数据。如果曾经较为深入地使用过 Microsoft Access，你就应该接触过 SQL 了。

业务逻辑是运营的大脑。所有复杂的运算都发生在这层，`IF {Charlie said no;} THEN {kill Charlie;}` 这样的语句也在这层执行。你的工程师将使用 Java、C++ 或其他类似语言来构建这一层。

展现层通常是 HTML 和 JavaScript。它将你的业务逻辑输出的数据按照一定的版式展现出来，这样数据会比较美观。JavaScript 允许用户进行实时交互。

AJAX（Asynchronous JavaScript and XML，异步 JavaScript 和 XML）则只是一个允许 JavaScript 向业务逻辑层提交迷你页面式的请求的技术，它不需要用户提交整个

表单。服务器也不再返回 HTML 和 JavaScript，它只返回一小串 XML 格式的数据[①]。所以 AJAX 丝毫不影响这种三层架构。

将三层架构压至两层也是可能的，只需允许工程团队编写一个文件来同时包含业务逻辑和展现逻辑即可。更令人不敢想象的是，一些框架甚至使用特别的方式与数据库建立连接，使数据库能返回供前端 JavaScript 直接使用的 XML 格式数据。这些杰出的时间节约工具初用起来是极好的，但会造成长期的麻烦。它们对内部项目非常有用，但不适用于对外销售的业务。

9.2　第二个 S：服务

面向服务的架构（SOA，Service-Oriented Architecture）与三层架构没有多大关系，但你会需要它的。SOA 将包含业务逻辑的中间层分解成一系列独立的服务。这些服务可能运行在相同的服务器上，但它们的构建、版本管理和运行都是独立的。图 9-2 展示了 SOA 的一个示例。

图 9-2 展示的 SOA 中，工程团队把理解查理回应的系统放在回应服务（answer service）中，根据查理的回应决定要做什么的逻辑被放在决策服务（decider service）中。

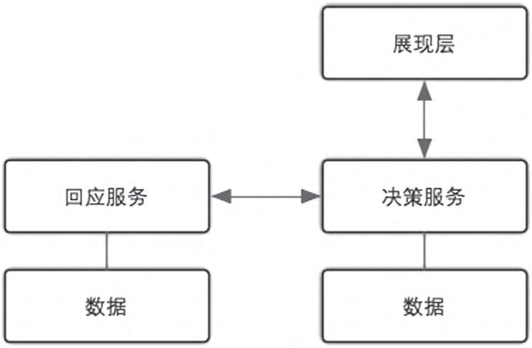


图 9-2　一个围绕服务组织的系统架构

①更流行的是 json 格式的数据。——译者注

这些服务通过应用程序编程接口（API，Application Programming Interface）连接。你可以认为他们和远程过程调用（PRC，Remote Procedure Call）是一个东西，尽管他们并不完全相同。所以 API 和 RPC 使得一个服务可以和另一个服务进行通信，不要让任何人愚弄你。决策服务会通过一个 API 来询问回应服务："查理的回应是什么？"这个 API 可能写成这个样子：

```
whatIsTheAnswer(Charlie)
```

回应服务将回应返回给决策服务，然后决策服务选择该对查理做什么事情。这些 API 都需要写进产品需求文档中。关于什么系统应该放在什么服务的边界并不特别重要。事实上，你甚至可以使用公司之外的服务，如用于结清交易的信用卡处理程序或存储数据的 Amazon S3。对于你的系统来说真正重要的东西是快速和可扩容。

如果你想要一个快速的系统，请尽可能避免服务链。我知道你也许认定服务可以拯救世界（当我在亚马逊的时候，他们确实就是这么做的），但请看图 9-3 所展示的 SOA。

在图 9-3 中，我们决定将一些我们真正关心的回应外包给一个新的魔法 8 号球服务。如果你对速度极其狂热，认为如果不能在 2 秒内提供响应用户就会离开，你就必须说服工程团队去满足 2 秒 99.9% 的服务水平协议（SLA，Service-Level Agreement）。这个 SLA 意味着只允许千分之一的响应超过 2 秒。听起来不错，不是吗？

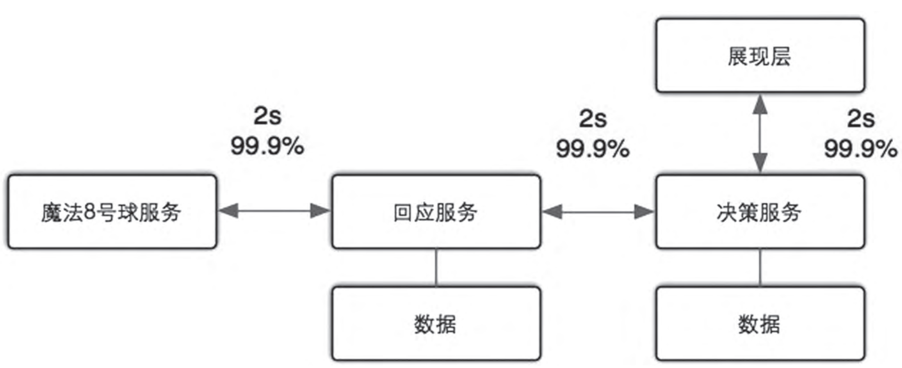


图 9-3 一个带有服务链的 SOA

如果性能只取决于魔法 8 号球服务那还好。但不幸的是，决策服务依赖于回应服务，这中间也有个 2 秒 SLA。因此一些用户不得不等待 6 秒才能获得响应，他们必须等待链中的每个服务返回结果。更糟糕的是，大部分系统传递响应的时间是呈正态分布的，这意味着你很可能看到一个接近 2 秒的用户平均响应时间，这只是平均值，不是最大值。所以尽量不要使用服务链，多去寻找能替代它的方案。

SOA 的缺陷

现谷歌软件工程师、前亚马逊高级工程师史蒂夫·耶奇曾写过一篇语气激昂的文章，论述为什么亚马逊内部的 SOA 从深层次上来讲要比谷歌无序的系统集成要好得多。他指出了一些值得关注的点，如下。

- 当你拥有大量的服务时，如果客户发现了一个问题，你可能不得不为了追踪客户看到的问题而去排查相当多的服务，直到你确定是哪个服务的问题。良好的监控能帮助缓解这个问题。
- 团队之间的依赖越来越成为一个问题，如果一个团队在没有通知你的情况下修改了他们的 API，你的系统就很容易遭到破坏。因此每个团队都必须确保向前兼容并主动通知客户，但这做起来很难。
- 构建一个出色的沙箱或测试环境并不容易，它要求你的每个系统都必须存在于沙箱中，这样才不会污染拥有海量数据的生产系统。但即便所有系统都在沙箱中，由于各个系统会频繁删除数据，你很难保证数据的一致性（如，订单和配送数据在订单服务和配送服务中是一致的）。

尽管 SOA 有这样的缺陷，史蒂夫和我依然相信，如果你想追求可扩容性（或称可伸缩性）、可扩展性（或称可升级性）以及其他优秀的特性，面向服务是一个值得遵循的好方法。

9.3 第三个 S：速度

我们已经证实了服务链会降低速度，也证实了面向服务的架构是优秀的，那么为什么不让你的两个服务在三层架构的顶层展现层连接呢，就像图 9-4 展示的那样？

你可以这样做，但一般情况下这不是个好主意。确实 AJAX 很擅长让 JavaScript

执行多个请求，也能把这个场景的 99.9% 最大延迟控制在 2 秒。但真正应用的时候那些预期在 JavaScript 中使用的 API 常常会显得脆弱。另外，大量的不同的依赖最终会体现在各种各样的 JavaScript 文件中，这样的系统很难管理。

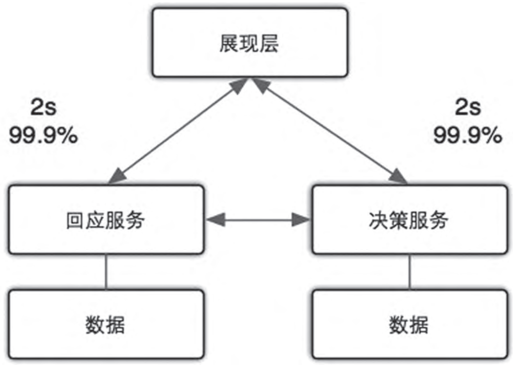


图 9-4 在展现层连接服务

为了应对这些复杂性管理带来的挑战，一个可行的工作方法是分开加载应用程序的各个独立模块。换句话说，如果有两个不相关的特性，你可以将它们的代码彻底分开，如分成两个独立的 JavaScript 文件，它们可以并行加载，也可以并行修改。文件和服务的独立性将帮助你快速扩容和修改你的服务。这种方法和 JavaScript 一次性加载所有东西的区别在于封装（Encapsulation）。你需要将全部功能封装在一起，它们才能各自独立工作。

缓存（Caching）是另一种解决服务链速度问题的方法。缓存是指数据源的一份副本。 你可以有页面的缓存，XML 的缓存或者硬盘的缓存。你能缓存任何东西。像 Akamai 这种内容分发网络无非就是靠缓存。缓存的一些有趣、有价值的属性使得你可以利用它来提升系统的速度。

缓存可以从它的后备存储器中复制所有数据，也可以只复制部分。一个缓存了所有数据的系统被称为“完整缓存”（cache complete），它拥有出色的灵活性，即便后备数据存储器（如一个关系型数据库）坏了，你依然可以从缓存中读到数据。

缓存的完整性很有价值，但有时你只需要部分缓存。例如，如果 90% 的请求都

只是针对某 10% 的数据，你可以部署 100 个小型缓存来存储那 10% 的数据，这样在 90% 的时间里你都拥有了堪比 100 台服务器的性能。因为缓存消耗的资源少，你仅需花费购买完整服务器所需金额的一小部分就能实现相同的性能提升。当你的服务需要的数据不在缓存中时，称为“缓存缺失”；优秀的缓存机制会“通读”后备存储器并传回数据，虽然这比从缓存中取要慢得多。拙劣的缓存机制则不会给你任何东西，这也正是它拙劣的原因。

聪明的缓存策略不只会通读后备存储器，它还会将读取到的数据缓存起来。这个方法可以促进缓存的完整性，或者起码可以通过保证一定量的请求来使正确的 10% 的数据存在缓存中。一个更为简单的方法是一边让通读缓存存下它读取到的数据，一边把最近使用最少的缓存条目清除出去，留出存放新数据的空间。按这种方式设计的缓存适用于博客这类重内容的网站，因为这些网站上新内容会得到最大的点击量，老内容则随着时间的流逝获得的流量越来越少。

缓存需要时间去更新。当你的数据存储中的某个值发生变化了，它必须先写入后备存储器，然后后备存储器必须更新所有的缓存。在一些设计拙劣的环境中，这会导致缓存的不一致性，对一个相同的东西用户会看到两个不同的值。如果你能建立用户和缓存的黏性，你就能实现通写缓存，让数据先写入缓存，再写入后备存储器。缓存要么通过通读来填满，要么通过“预热”来填满。空的通读缓存在数据第一次被请求时性能表现极差，如果不想忍受，你需要预先将数据填入缓存，有时这个过程被叫做“预热”缓存。你需要单独编写软件来做这件事情。

以上只是对缓存相对粗略的说明，不过好消息是，你并不需要知道关于缓存的所有知识就能明白它是非常重要的。你现在的知识已经够让你向工程团队就缓存机制提出有意义的问题了，或者至少能让你在发现没有缓存机制时产生一丝疑惑。

9.4　第四个 S：扩容

然而有时只有缓存还不够。当获得更多用户时，你需要系统不仅能检索更多的数据，还要能做更多的思考。这意味着你需要准备更多的服务器。使用 Amazon S3 和 Google AppEngine 等第三方主机托管服务的一个最大的好处是，他们能够帮你解决大量这类问题。如果你没有构建在这些系统上，请继续往下看。

在可以通过新增服务器来实现扩容之前，需要理解所有服务器之所以在用户看来只有一个，是因为他们都被置于虚拟 IP（VIP）之下。VIP 允许使用单个互联网地址来表示拥有的所有服务器。VIP 地址是通过一种精妙的可以看做是硬件的东西来管理的，这种硬件为每个用户分配一个空闲的服务器并让他们停留在那里。你可以购买或租借 VIP 硬件。它们非常非常的昂贵，但你不能不用。或者你可以遵循我之前的建议，使用完整的托管主机集群来彻底避免这种复杂性。现在你可以通过新增服务器来扩容了，你接下来一定会听到系统能“线性”或“纵向”扩容之类的词。真正的极客可能会说“常数时间”或“N”。它们都是一个意思（记住“常数时间”，用这个词代表“我可比你聪明噢”）。你能通过增加服务器来增加容量，且增加的每个服务器都能带来接近完整服务器的容量。不过在许多系统中你依然会遇到瓶颈，如在 VIP 这块，一个 VIP 只支持那么多连接。这时面向服务的架构的一个好处是你可以独立为每个服务扩容，当决策服务需要更多马力，回应服务需要更多空间时，你可以通过合理分配相应类型的硬件给他们扩容。

要想这种支持扩容的设计正常工作，你必须将数据存储起来以便它能快捷延伸至新增的服务器。创建一个算法来做这个会比较棘手。比如以人名为例。你可以将人名根据首字母分别保存在 26 个服务器中，但这套方案不靠谱，因为当用户查询“Smith”时会一直不停地请求 S 服务器，而 X、Y、Z 服务器则处于空闲状态。幸运的是还有一些不那么幼稚的方法，只需确保你的工程团队采用其中一种即可。一个稍稍能减少麻烦的方法是为每个用户创建一个单调增长的客户 ID，然后根据客户 ID 存储数据。这样你就能够通过增加服务器来应对日益增长的客户了。

当然，关于跨服务器存储数据的方法有很多，你可以借此更好地实现扩容。以数据库为例。依靠单个数据库支持的系统无法纵向扩容，因为它们假定了所有的数据都存放在一个地方，而不是分布在多个服务器上。因此，如果想要实现扩容，你需要买一个更大的服务器来取代之前的服务器，这意味着需要签张大额支票给 Oracle 以及提升在 Dell 的信用上限。如果拥有一个这样的系统，你很可能会到达这样一个节点：你拥有的数据超过了拥有的金钱，这时候结果不再会被缓存了，你的速度会很快慢下来。这样的系统扩容成本可算是呈指数级上升了，要是真搁在你的头上简直就是一场灾难。

如果陷入了这样的困境，也许需要去寻找 NoSQL 一类的数据库技术，这类数据

库系统先天就能很好支持数据跨服务器分布。但是由于你并没有设计完整的存储基础设施，无法真正地定义数据该存哪以及查询该往哪走。这个意思是说，当数据库中的数据发生变化时，这个变化必须传送至其他服务器，而如果在传送间隙中做了两次查询，你可能得到不一致的结果。或者换句话说，如果你更新了系统可能看到了我欠你5元，但我可能看到的是我没欠你任何钱。这虽然不好，但它能够快速返回给你结果并最终完成数据的传送。我们把这个叫做*最终一致性*。最终一致性可能适用于简历的更新，但未必适用于在扑克游戏中记录胜负。

你还可以通过创建*索引*来解决某些类型的性能瓶颈。索引通过采用不同于存储数据所用的描述数据的方式来帮助你快速找到所需要的数据，例如，假设一个用户想快速搜索“罗格·史密斯”，但服务器上的数据是按照客户ID组织的。如果没有索引，你将不得不遍历所有记录，极其缺乏效率。这种方式被称为“*全表扫描*”，如果你的系统是这样的，应该开始着手寻找一个新的工程主管。如果有索引，你将拥有一张按照名称排列的用户列表，可以直接跳到“罗格·史密斯”一栏查找他的相关信息。索引不是免费的午餐，你必须存储并更新它们，但它们的价值抵得上日常运维的成本。

9.5 如何询问正确的技术问题

仅仅看过几页架构相关的内容并不能让你获得计算机科学学位。不要仅仅根据本书的说明就试图去设计系统，你会伤害到自己和其他人的。但你现在的知识已经足够向工程团队询问一些重要的问题并听懂他们大部分的回答了。那些还听不懂的部分则很可能是*星球大战*的黑话。你必须询问一些这类问题，它们会揭示出一些潜藏的问题并帮助你的团队想清楚他们的设计。你也许相信团队已经想清楚设计了，但考虑到他们眼中的“设计”是什么样，你会大吃一惊的。以下是一些可以询问的问题。

能请你给我画张系统图吗？

你的目的是理解系统图中所有方框都是干什么的。先从离客户最近的方框入手，询问它存放了什么数据，它干了什么，它接收和发送了什么数据。按照你的方式询问所有方框的情况，直到了解了它们都是干什么的。寻找我们这章讨论过的东西，如服务分离、服务链、索引和扩容。

结果从这个方框传到那个方框的延迟是多少？

你应该能通过通览系统图来识别出服务链，询问这类设计的必要性，并弄清楚整体响应时间是多少和最坏的情况下是多少。如果你发现图中有个地方的响应延迟非常长，询问怎样才能通过缓存、纵向扩容该服务，或将部分逻辑分到其他服务中去来提升它的性能。

可以扩容到 *N* 吗？

考虑到这本书将有很大概率帮助你取得无比巨大的成功，因此你必须扩容到的 *N* 也将是一个无比巨大的数字。想想当你使这个数字变得无比巨大时会发生什么。我把“无比巨大”定义为每秒有 1 万个请求，或者 1 亿条客户记录，抑或每天 1 百万个订单。那么你的工程团队需要干什么？增加更多的 A 方框就够了吗，还是需要打电话给 Oracle 并让老板签下一张无比肉痛的支票呢？

去掉 B 方框有什么影响？

了解你的系统包括了解它会如何出错并如何恢复（希望它能恢复）。确保自己了解到了系统的哪些部分会产生致命的错误，并让工程团队优先考虑对保证这部分的稳定性进行投入。

我们是围绕组织边界或系统边界来架构吗？

有时你会发现 SOA 反映的是公司的运营方式，而不是数据或应用程序的结构方式。而由于公司的组织结构不是遵照摩尔定律设计的，所以要避免这样的设计。当出现一些团队难以协作而导致多余的或不正常的系统被构建出来时，检查上面这一点并主动应对。

能通过缓存什么数据来提升性能吗？

我们花了大量时间在缓存上，因为它们很重要。它们能提升性能，提升稳健性并降低运维成本。识别静态数据和常规查询并讨论如何缓存它们。不要忘记询问关于缓存完整性的问题。

能通过独立加载什么数据来提升性能吗？

就像我们讨论过将回答服务和决策服务独立开来分别返回结果一样，你也应该询问系统的某些部分是否可以分离。例如，如果你能够单独加载广告，那么那些系统就可以完全独立运行，你的整个系统就更有弹性，即便广告系统坏了，用户也可以完成他们的首要任务。

第 10 章 胜在沟通 10

SHIPPING GREATNESS

如果正致力于软件交付，几乎肯定会有海量的信息需要传达，海量的状态需要收集，海量的检查需要执行，还有海量的其他细节需要担忧。你会有大量的邮件需要发出以及大量的会议需要组织。这都不是什么好消息，但这正是公司发你工资的原因。好消息是：只要你懂一点技巧，掌握以上任何一件事情都不是难事。

其中一个关键的技巧是尽可能少开会，但不要不开会。很多时候通过一封优质的邮件就能完全避免开会。我们就先从这开始吧。我一直对一个现象感到很惊讶：谷歌和亚马逊的副总裁们都很擅长写邮件，但他们底下那些缺乏经验的团队主管却写得很糟糕。而你，等到读完本章，最起码也可以认为自己做好成为副总裁的准备了。

10.1 如何写好邮件

爱因斯坦曾说：“如果不能将事物简单地表达出来，你就没有真正地理解它。”当我收到一封长篇大论讨论某个简单问题的邮件时，我会直接把它存档。“长邮件综合征”太普遍了，工程师们为了回复这类邮件而专门发明了一个简写方式：“太长没读（tl;dr）”，意思是说“太长了，没法读（too long;didn’t read）”。可见不是每个人都能写好邮件的。亚马逊首位项目集经理、前副总裁金・雷切米勒曾在挑选一位项目集经理到她的团队时说：“她的邮件简直就是清爽的代表。”这是来自亚马逊高层很高的评价。现在亚马逊要求所有负责带人的经理候选人都需要在面试时提交一份写作样本。可见写好邮件是多么重要。

作为一位出色的领导者，如果想让上司更认可你并获得更高的薪水，你需要不断向团队传递清晰的、具体的信息，使他们明确方向、坚守使命。你还必须做好向上管理，这意味着你要向比你更忙的、有更多邮件需要处理的人们传达与决策或进展相关的大小事项。因此写好邮件对你能否成功至关重要。写邮件最主要的目标应该是清晰、简要地传递单个信息。在亚马逊，人们普遍使用“清爽”这个词来定义清晰、简要的信息。注意到没有，亚马逊正试着用一个词来代表两个词（清晰、简要）——行动胜于千言！然而在谷歌，清爽的信息在个体贡献者中推行得并不是很好，因为这里

的文化带点消极反叛，这时候你需要快速切换方式以迎合受众。你应从始至终精心撰写你的邮件，使其短小、具体、详实且清晰传递了单个信息。为了达成这些目标，我试着像记者写新闻一样写邮件。

10.1.1　像记者写新闻一样写邮件

优秀的记者会将他们想表达的最重要的事情放在文章开头。一篇《华尔街日报》的报道可能这样开头："经济依然糟糕。"而你则更可能以"我们无法按期发布"作为邮件开头。接下来是一些说明原因的句子，尽可能让它清晰，如："因为我们的两个依赖方没有按期准备就绪。"拙劣的邮件会将理由和辩解放在前面，匆匆扫过的阅读者不知道你在为什么而道歉。例如，拙劣的撰写人会这样写：

我们发现名称查找服务直到14号才能准备就绪。除此之外，我们团队的两个成员（查理和萨莎）都得了流感，因此我们失去了两个星期的生产力。由于这些并非我们过错的阻碍，我们不太可能按期发布。

优秀的撰写人则会这样写：

汤姆和杰瑞，

我们必须将发布日期延后2周，即从8月7日延后至8月21日。我们不得不这样做的原因是：

- 工程团队有人生病，开发进度受到了影响；
- 我们所依赖的名称查找服务要到8月14日才能准备就绪。

祝好！

克里斯

在拙劣的发件人写就的邮件中，撰写人"主次不分"，这种说法来自新闻行业，因为这种现象在新闻行业太普遍了。优秀的记者绝对不会犯这个错误，因为这会让他非常窘迫。在这个拙劣的邮件的例子中，阅读者很容易在看过第一行后说："噢，日期推迟到了14号。"但实际不是推迟到14号，是要推迟2个星期，因为这个团队失去了两个星期的生产力。

优秀的发件人则没有犯这种错误，他还多用了20秒时间来为邮件加上称呼、敬语和署名，这样可以帮助信息传递给正确的受众（特别是邮件还抄送给一些人的时候），还可以帮助减少潜在的恶意。我没有任何证据来支持这套理论，但它又没有任

何害处，何乐而不为呢？署名还能让阅读者在 iPhone 上阅读时无须拉回顶部即可知道信息是谁发的。

优秀的发件人还有个最为出色的地方在于使用了*精确增量表达法*。我喜欢这样的发件人！

10.1.2 使用精确增量表达法

精确增量表达法是一种让数字更易被理解的技巧，适合应对那些超快速阅读的人们。使用它很简单，只需将你的数字改成下面的格式：

将某个变量增加或减少 XXX(差值)，即从 XXX(开始值)增加或减少到 XXX(结束值)。

这个格式让阅读者知道发生了什么事情，即某个变量将*增加或减少*。那么多少呢？见*差值*。那之前是多少呢？见*开始值*。如果阅读者真正关心的只是最后的交付日期呢？他可以简单地跳去看*结束值*。注意到优秀的发件人正是按照精确增量表达法来表达他的更新的：

*我们必须将发布日期**延后** 2 周，即从 8 月 7 日延后至 8 月 21 日。*

如果你想试着做得更好，可以给目标加上开始时间、结束时间、总持续时间等基于时间的要素。新的格式将是这样：

在 XXX（开始时间）到 XXX（结束时间）这 XXX（总持续时间）时间内，将某个变量增加或减少 XXX（差值），即从 XXX（开始值）到 XXX（结束值）。

使用这种格式能让任何人都能快速看懂你做了什么，将产生多大影响，你将何时开始，何时可与你确认是否完成。你仅通过一句话就可以无歧义地表达所有事情。精确增量表达法就是这样一个能极大提升明晰度的有力工具。但它也不容易学会，所以在刚开始实践的时候会不舒服是正常的，不要气馁。它是那么简洁又那么强大，你值得努力去掌握。

10.1.3 分点阐释原因

优秀的发件人将他的逻辑依据从视觉上划分成多个点进行阐释，从而人们能够轻松发现要点。你的团队可能想阅读全部的逻辑依据，而 VP 若是对你信任有加，他会略过它们而只阅读你邮件的第一行。幸运的是，你现在是一个优秀的发件人，你的第

一行已经优先传递了最重要的信息。

如果你无法分点写出逻辑依据，而且原因还是你不知道要传递的关键信息背后的缘由是什么，那你的问题就大了。你最好不要发这个邮件，没有比冒然推迟日期但不知道为什么更白痴的行为了。如果你必须提供某种近况的更新，那么依靠你可悲而浅薄的知识勉强说些什么吧，但不要说些莫须有的东西。自愿承受后果并再承诺一个有事实依据的时间吧。

顺便说一下：在优秀的发件人写就的邮件中，各点之间的行间距是很有意义的。适当留白能帮助你清晰分隔出邮件中重要或特殊的部分。

10.1.4　立即停笔，你已经写完了这封邮件

到这里你的邮件就写完了。或许你可以给你的逻辑依据再添加一点数据，或许你想添加一个打开状态面板或项目计划的链接来炫耀逻辑依据是多么真实。但真的不需要这样做，忙碌的高管会在他的手机上阅读这封简单的邮件，然后再读下一封。这就是你要达到的目标，所以立即停笔并开始下一项任务吧。

10.1.5　设法用建议取代质疑

当你对工作还不熟时，会发出大量充斥着问题的邮件，如：“为什么保存按钮是红色的？”“为什么工作量都还没估完就确定了发布日期？”“为什么我们把时间浪费在嚼口香糖上？”

这些实际上都不算问题，虽然标点符号会让你这样觉得。相反，它们听起来像是说“你们这帮家伙是设计白痴”“你知道一点关于项目管理的知识吗”以及“为什么有人要嚼口香糖”。更糟糕的是在大多数场合下一旦你开始问这些问题，团队会回给你各种混杂的信号。你会听到类似于这样的话：“这些都是好问题，只是你最好换种语气问他们”和“大家都被你问得糊涂了”。

问“为什么工作量都还没估完就确定了发布日期”是一种带有倾向性或引导性的质疑。它断言了这样的观点：在没弄清楚你得做多少工作之前就定下发布日期是愚蠢的。当然这样很愚蠢！但你怎么知道不是因为合同期限或其他外力导致的呢？所以最起码试着把这种倾向性从你的问题中剥离掉。这样做对你只有好处，没有坏处。

当换成从人类动力学角度来看这些问题时，我们会发现一个奇特的地方，即有时

候（但不总是）建议会比质疑更容易被接受。建议能让他人挑挑刺，而质疑是直接冲着他人的，会加剧他人的抵触情绪。也许有分析研究证明这一点，但我不能明显地感觉到这种差别，你也不一定能从收到的反馈中明显感觉到它。所以仅作为一个实验，不妨试下这种建议式问法。

■ 能把保存按钮改为蓝色吗？

■ 发布日期还可以协商吗？

■ 我能嚼个口香糖吗？

这些建议可能比一开始那些直接的质疑更让你我恼火，但你可能会发现面对新团队时，这些问题更容易问出口。

10.1.6 考虑受众的感受

考虑他人的感受不是件轻松的事，但有时候很有必要。高级管理者向一线员工写邮件时就得多花点笔墨以照顾员工的感受。就我个人而言，我非常注重邮件的清爽性，所以常常会直截了当地传达决定和逻辑依据给团队的个体贡献者们，有时候他们并不会友好地接受。你懂的，什么样的员工就会用什么样的方式作出反应。

我把这种传达上的失败归因于没有考虑受众的感受。对于很敏感的团队，如那些花了一年时间做某个产品而你却打算砍掉它的人们，最好不要这样写：

小弟，

你被解雇了。这是我们为了达到减员 10%（从 100 名减到 90 名）的目标而做出的决定之一。我们选择你是因为：

■ 我们不喜欢你穿的那件衬衫——你懂的，就是跳豆花舞时穿的那件。

■ 你的工程经理没有力挺你。

■ 总得有人被解雇。

希望你能有个好心情。

你的老板

这样一封邮件即便再遵循那些我所赞赏的最佳实践，人们也不太可能安心接受。就我个人而言，我是希望更多人去写这种直率的、组织良好的邮件的，因为与其和被动攻击式邮件催生的存在性焦虑做斗争，还是改变个人的风格更为容易一些。但是，我们要友善对人……

各位团队成员，

我们是一支由100位一流员工组成的梦之队。在过去一年里，你们中的许多人——事实上是最最重要的10%，为我们非常看重的一个系统日以继夜地工作。就像软件中的大多数东西都会快速发生变化一样，我们最近才意识到对于沃尔夫冈系统的需求已经发生变化了，我们需要将业务转向能更好利用我们优势的地方。这意味着我们将在今天停止对该项目的投入。这个改变对不同的人有不同的意义，但它的要点在于为沃尔夫冈项目工作的团队将不得不寻找新的机会。你们应尽快与工程经理进行1对1的交流。如果有任何问题，请直接发邮件给我或者你们的经理，不要有什么顾虑。

克里斯

好吧，我有点言过其实了，事实上高管从不会关心你穿什么。但一般来说不管公司规模大小，不管什么时候你都会看到这种邮件。我“真的”看不下去这种起抚慰作用的邮件，也弄不懂它实际想说什么。但许多阅读者喜欢这种长篇大论的、善解人意的邮件，它合理地解释了变化，称赞了做事的人，并把变化说成是所有人的事情。它还使用了一些重要的时髦的字眼，如“转向”、“变化”、“快速”等。

我想完全不理会这种善解人意的邮件，但我不能。语境真的很重要。如果你作为副总裁写邮件给另外一个副总裁，你可以将邮件写得很棒。如果你作为一线的产品经理或技术主管写邮件给你的经理们，你必须将邮件写得很棒。如果你作为高管或副总裁写邮件给你的下属，你则必须将邮件写得善解人意。如果你不根据受众适当调整邮件风格，上到副总裁，下到个体贡献者，你都免不了要得罪。

10.2　如何应对五种类型的会议

如果邮件不管用，那可能是到了要开会的时候。开会可能会痛苦，也可能会高效甚至快乐（是的，你能让开会变得快乐）。让开会变得快乐的最佳途径是理解每种会议的结构、目的和效果。下面有五种你需要掌握的会议类型，排名不分先后。

1. 团队会议

这类会议用来了解近况以及利用团队合力来深入讨论和解决特定问题。虽然团队会议中解决的大多数问题理论上通过邮件也能解决，但只是理论上而已，所以你还是需要这种会议来承担这些工作。

2. 站会

站会是一种超级简短的会议，它只用来交流近况，促使团队内部信息透明、责任到位。在会议中每个人都站着，这样可以帮助保持会议的简短。

3. 1 对 1

1 对 1 会议是指只有你和另外一个人之间的会议。这类会议或许是最值得开的，因为在会议中你们能坦率地交谈。而且会议也给了你们专门时间来完成需要互相协作的任务。

4. 产品 / 工程 / 用户体验评审

这是一种大规模会议，通常会有一些大老板参加。这个会议既要向高管通报产品进展，又要收集组织内最富有经验的人们的反馈建议。团队需要仔细准备这类会议，如果搞砸了，项目势必被要求重来，所以从这个角度来说这类会议是相当昂贵的。

5. 头脑风暴会

这是所有会议中最有趣的，它形式自由，能激发想法，还能让团队主动参与到问题的解决中去。

10.2.1 团队会议

团队会议每周开一次，每次 30~60 分钟，你和你的工程团队需要参加，一般由你主持。如果团队中有技术主管或高级工程师乐意主持会议，当然可以，交给他吧！这类会议的目的是促使团队坚守使命并就当前一些悬而未决的问题达成共识。议事日程需要提前发布。等到开发快要结束时，促使坚守使命和达成广泛共识通常就没那么重要了，你可以考虑取消这类会议并改成开站会，取消的最佳时机是你开始跟踪 Bug 燃尽图的时候。

当你开始团队会议的时候，一个不错的开场方法是先回顾你的数据指标。你需要知道产品或开发有什么进展。它们有任何中断或显著变化吗？例如，鲍勃是不是花了整整一个礼拜的时间在家陪他的猫？或者萨里是不是发现了一些不得不尽快解决的隐私问题？除此之外，你的时间表上的事情完成得怎么样了？通常这个时候大家需要打开笔记本，更新项目跟进表中的各个相关字段。

如果第一次在会上发现有人在走查电子表格并一边编辑它一边大声说出来，你可能会想："太疯狂了，我们应该提前把这个事情做完。"你是对的，这是最理想的做

法，但不现实。鉴于工程师完全有能力做任何事，如代码评审、编写测试或浏览xkcd.com，所以他宁愿去做这些事情，而不是更新团队电子表格。在团队会议中，这个工程师就像笼中鸟，干不了别的了，于是他就有闲情更新表格了。除此之外，在讨论细节时，工程师可能会解释为什么一些任务花费的时间比预期更长或更短。这些解释很有用，因为其他团队成员可能不知道你搭建的系统是这么扯淡还大小写敏感，这只是一个例子。要知道在现实生活中越不想其发生的事情越可能发生。

你还可以利用团队会议来达到更高层次的目的。在季度开始，我会重新审视团队季度目标，并调整目标直到所有人都认同。在季度中间，团队会议可以确保你不会在方向盘上睡觉以至于团队状态“哗”地从绿到红，或许至少可以让团队状态先从绿到黄，再从黄到红。在季度结束，我们会在团队会议上评估我们相对于目标的进展情况。如果你遵循了这个过程，你的团队将能始终把控未来，坚守项目目标，专注做正确的事。

当项目计划表更新完毕，即会议的信息更新部分结束后，你或许需要聊下所有需要团队成员知晓的最新进展，这些重要进展包括行业变化、业务近况和其他与使命相关的事项。切记，确保团队坚守使命是你的工作。

会议的最后部分专用于讨论并解决那些悬而未决的问题。在会议开始时收集需要团队讨论的问题清单，再在信息更新部分结束后逐一讨论清楚。做好记录，指派任务，然后处理下一个问题。处理完这些问题后，问下团队是否还有其他的东西需要讨论。等 7 秒看下是否有人回应。7 秒的安静时间似乎很长，让人感觉颇不自在，但它却足以让你团队中最内向的人也可以无保留地发表意见。总而言之，开团队会议比不开好，哪怕这个团队会议超级简短。它使得团队有机会以一种不同的方式聚集在一起，它还提供了喘息的时间。因此即便我认为没有什么东西需要讨论，我也会召集一个简短的会议来和团队碰一下。我发现在那安静的 7 秒钟去凝视团队的不同成员，会经常挖掘出令我吃惊的东西。

10.2.2　站会

Scrum 和敏捷开发的拥护者都信赖每日站会。最有成效的站会的确要求每个人都站着。之所以存在这样一个真实的站立着的会议，是因为它可以使人们离开他们的电脑并站起来，这样他们就没那么舒服，会议也就不得不尽快开完。我喜欢每天开一次站

会，因为这能促使团队内部责任到位、信息透明。每个人在站会中应该汇报下列信息：

- 我昨天做了什么
- 我今天要做什么
- 我是否遇到阻碍

如果你照做了，整个团队将很清楚每个人的表现和项目的状态。阻碍项目的问题将很快浮出水面。更为重要的是，每个人遇到的挑战常常能得到其他人的帮助。例如，当珍妮说："我一直在搞这个构建，我无法让测试 X 通过。"这时通常会冒一个同事肖恩说："是的，我上周也遇到了这个问题，会后我和你聊下这个问题，也许我能帮上忙。"长期下来，这种类型的信息互通能节约大量时间。

我在谷歌和亚马逊的团队都喜欢在 11:30 或 12:00 的时候开站会，时间不超过 15 分钟。这个时间点刚好在午饭前，团队成员都处在临界点上，所以不太可能会滔滔不绝。会议中还需要有人（比如项目集经理或开发主管）留意讨论是否陷入细节，这很关键。例如，当肖恩和珍妮开始讨论起为什么测试 X 无法通过时，有人需要说，"嘿，你们能私下聊么？"这样站会就能继续开下去并保持简短。

一个 10 人的站会可以在 10 分钟内开完。我曾在谷歌见过 50 人的站会花了不到 20 分钟就开完了，但他们主要是重申目标，而不是收集近况。

我不赞成 Scrum 中区分猪和鸡的做法，它只让猪（即开发者）说话。我提倡信息透明，我喜欢让产品经理、项目经理、设计师、用户体验研究员甚至实习生都来参加站会，只要他们时间允许。我的工程团队成员喜欢听关于业务拓展近况的 30 秒简述，这帮助他们巩固了项目不仅仅只有他们在努力的想法。

我还喜欢让开发主管就项目状态做 30 秒的简报。这个方法很好，它为会议设定背景，并促使团队回忆起一些重要的事情。下面是开发主管介绍最新状态的一个示例，你可以依葫芦画瓢：

> 今天有 30 个 Bug，我们的发现 / 解决率终于开始下降了，因此按原定计划在下周末发布内部测试版，所以请赶在周二结束前完成修改。切记，周二结束前。另外我这周轮值，我的生产率有可能下降。

10.2.3 1 对 1 会议

1 对 1 会议非常适合主管们相约在一起坦率地进行交流并公开问题。它小而专注

的特点也使得它很适合用来完成需要合作的工作。当你在 1 对 1 会议上和对方一致同意发一封邮件时，你可以轻松花 2 分钟时间把邮件发出去，而不需要记下来并等到会后再去执行。这种做法能让你不用会后再花时间回想当时做决定时的背景，还能减少这项任务未完成的几率。同样，你和你的市场主管也能在 30 分钟的 1 对 1 会议上花 15 分钟来共同修订博客文章的第一段。在会议过程中完成工作是 1 对 1 会议的精髓之一。

请安排每周或每两周与所有主管开一轮 1 对 1 的会议，即便你觉得没有必要。1 对 1 会议时间的长短取决于你需要处理什么事情。30 分钟是一个不错的起步时间，50 分钟也不是不可能。你可能会议中间发现一些你并不知道但也需要讨论的事情。如果几周过后会议不再富有成效，你可以取消它，或者改为每月一次。

10.2.4 产品、用户体验和工程设计评审会

任何种类的“评审会”通常都是有大老板参加的大规模会议。其中产品评审会的首要目标是让老板们认可你的产品方向，提供给你反馈，或让他们知道你的最新进展。后面“如何做好演示”一节中的所有建议也适用于评审会。你要先想清楚你要传递的确切信息是什么，然后尽可能清晰、简洁地传递出去。由于这些会议的参与者通常都很忙，最好将会议时间控制在 30 分钟以内。

用户体验评审会则只需要准备产品原型图就可以了。最好让设计师演示他们自己制作的原型。在设计师演示之前，你应花一小段时间为演示做一下铺垫，讲一下为什么你在这儿，你的用户是谁，你的业务目标又是什么。你也可以亲自来演示，但前提是你的设计师是一个极其内敛的人，且过往经验表示你更擅长这种面向更高层次的走查演示。

工程评审会的目标是：赋予开发主管权力，收集周围经验最丰富的工程师的专业反馈，以及尽量少花时间在演示上。请提前审读材料，以免开发团队提出一些使命之外或与策略相悖的东西。如果将使命和策略成功传达给了团队，你就不会有任何问题。如果你不是负责演示的工程主管，则需要在评审过程时应对高官们踢过来的弧线球：“等等，为什么我们要构建 X 特性呢？”通常这些问题最好让工程主管来回答，因为回答这些问题能让他感觉自己被赋予了权力，而这正是你的目标之一。但这些问题常常会令你的工程主管措手不及，这时候你要赶紧补位。你肯定不想让他们认为你的团队对未来一无所知吧。

10.2.5 头脑风暴会

最后也是最有趣的一类会议是头脑风暴会。它的目标是收集尽可能多的想法，不管是新的产品名称，扩容问题的解决方案，还是系统错误的可能的根本原因，它都收集。头脑风暴的时长没有限制，但最好隔一个半小时休息一下。头脑风暴可以充满快乐和自由，但如果你想让它发挥更好的作用，还是需要设立一些基本规则的。我会在头脑风暴会上遵循 4 个基本规则，我确信它们能为会议带来更多创造性的效果。这些规则是：

- 不要在头脑风暴过程中批评他人的想法
- 说“是的，嗯……”
- 通过结构化来促进讨论
- 在头脑风暴结束时明确告知大家头脑风暴结束了

1. 不要在头脑风暴过程中批评他人的想法

最有效地践踏创意的方式是批评它们，然后会议室里就会多一个一声不吭的傻瓜，而你将少一个能贡献想法的人才。在会议一开始就讲清楚这条规则。还有一个有助于避免批评并鼓励人们贡献想法的方法，是将所有想法都写在白板上。大幅的画画在专用画纸上也可以，会议结束后你还可以带走它们。

2. 说“是的，嗯……”

即兴表演中有一个演员们制定（也可能是在面对大量嘘声中逐渐形成的）的用于避免情节停顿的规则。它叫“是的，嗯……”规则，说的是不管你的表演搭档说什么，你都回复“是的，嗯……”即使你的搭档说：“啊，你的裤子着火啦！”你也得说：“是的，嗯……都怪我碰到打火机了！我为什么会把打火机放在屁股口袋里？”

这个技巧在商业环境中也有着惊人的效果。它将创意带到下一个层次。鼓励你的团队使用这个技巧。

3. 通过结构化来促进讨论

威廉·爱德华兹·戴明是商业社会的一个传奇，他将创新过程带入工业领域。尽管他是一个保守、无趣的家伙，但他却信奉头脑风暴。他使用鱼骨图（很快我们就会详细介绍它）来记录和指导这类会议。图中每根鱼脊都代表一个他提出的问题，然后他的团队会头脑风暴出该问题的所有可能的答案。当他们抛完想法后，会继续头脑风

暴出所有答案背后可能的原因，如此一而再，再而三，直到他们得出所有问题的根本原因，或者感到厌倦。

结构化地讨论更有条理并能鼓励创造性。你不一定要用鱼骨图，有时候甚至只要定下三个需要解决的问题并设法解决它们，就能帮助你团队中更专业、更有条理的成员参与到你的过程中来。

4. 在头脑风暴结束时明确告知大家头脑风暴结束了

头脑风暴很有趣，但你也不能做一整天。到了某些点，你不得不结束头脑风暴并进入批判性分析阶段。切换模式时一定要和团队讲清楚，如果你没讲清楚，人们可能会随机跳回纯创意模式，当你努力将会议拉回正轨时，那些提出新想法的人们又会感到被拒绝。

爱德华·德·博诺在他的《六顶思考帽》一书中论述了该如何转变思考模式。他主张人们应该脱掉“绿色”的创意帽并戴上“黑色”的批判帽来分析这些选择。他所谓的换帽子是指你需要从一种思考方式猛地转变成另一种思考方式并对转变直言不讳。他还描述了其他四种帽子，如果你想进一步拓展思维，不妨去了解下。

10.3　如何组织好会议

每个领导者组织会议的方式或多或少有些不同。会议常常是你个人风格的表现，这意味着没有哪种最优的方法能组织好所有会议。然而有些最佳实践能让你组织的每个会议都更好一些，且不限于你的个人风格和组织的会议类型。如第 11 章关于如何处理冲突的最佳实践就适用于处理会议中的冲突。除去这个，我还遵循四个最佳实践：

- 会后立即发出主题纪要
- 允许改变开会的目的
- 拒绝在团队会议中发泄负面情绪，但允许在 1 对 1 会议中发泄
- 使用鱼骨图等辅助工具解决问题

10.3.1　会后立即发出主题纪要

彼得·德鲁克是一名执行绩效专家，著有《卓有成效的管理者》一书。他倡导在会议结束后应向所有相关人发送会议纪要。谷歌负责管理所有产品经理的前高级副总

裁乔纳森·罗森伯格也极力主张会后应发送会议纪要并抄送给他，就我而言，凡超过5人参加的会议我都会把会议纪要抄送给他。在他手下待过的产品经理没有一个能弄明白他是如何应付收件箱里的海量邮件的。

你之所以需要在会后立即发出会议纪要，是因为只有这样它的作用才能最大化，你的团队才能感觉到他们是会议的参与者和下一步计划的执行者。不要过于担心纪要的准确性，人们会纠正你的错误的。在纪要中先放结论和下一步计划，再放讨论的详情，这样那些不认同结论的阅读者也能了解你们得出它们的原因。这种组织纪要的方式应该是不言而喻的，但我在谷歌看到的所有来自产品经理的纪要都只有详情，让人难以卒读。请写一段清爽的，2~5行的概要放在纪要顶部，人们会感谢你的。难道你不认为这值得被感谢吗？

10.3.2 允许改变开会的目的

开会有三大目的：解决问题、收集信息、传递信息。一般会议都只为其中一到两个目的。有时候每周一次的团队会议能同时达到这三大目的，不过前提是你肯将它逐步完善。

开会目的是可以改变的，你千万要记住这点。当主持一个评审会那样的大型会议时，即使你的目的是传递信息（你会这样说“这是我们的计划……”），你也可能因为某种原因而不得不快速把目的转变成收集信息。其原因可能是你依赖的某个假设产生了变化。不妨遵循本章后面“如何做好演示”一节中关于如何转变、认真倾听和收集信息的建议。

当你在团队会议中发现问题时，接受它，无论它是团队动力问题、系统设计问题还是新需求。记得第5章所说的“坏的消息就是好的消息”，正面接受坏的消息并着手处理它。我认为你应该马上将坏消息记到会议纪要中去。如果你担心高管容易因此惊慌，你可以写成“团队正在积极调查这个问题”。不过确实，大部分高管都容易惊慌。

当会议中出现坏消息时，你需要扮演建导的角色。例如你必须首先认识到你的信息传达会议现在需要改为专注于问题解决了，将这个告诉你的团队，帮助他们将焦点转到新的目标上来。你可以说：“让我们暂停一下，先把这个问题解决了，可以吗？”你也可以先承认问题的存在，然后确定负责人和下一步计划，让这个问题在会后被妥善解决。你的团队会告诉你他们喜欢哪种方式，作为团队的服务者，你应在极大程度

上听从他们的指示。有些团队会不计一切地避免冲突，你需要为他们创造一个免受攻击的空间来解决问题，这就是建导。

10.3.3 拒绝在团队会议中发泄负面情绪，但允许在1对1会议中发泄

我坚信几乎所有的危机都是教育的机会。比如说有团队成员开始抱怨的时候就是一个机会。和团队会议不同，1对1会议则很适合个人发泄情绪。在这小小的舞台上，你可以化身为共鸣板，使用基本的主动式倾听的技巧来帮助你的成员说出他们的心声，比如说：“我听到你在说……”但当某人在团队会议中散发出强烈的负面情绪并完全不顾及他人或产品时，你需要严厉批评这种态度。长期弥漫着负面情绪会伤害你的团队成员，所以不妨借这个危机去教育你的团队，在面对问题时不要抱怨，而是要寻找建设性的应对方法。比如你可以向他们指出抱怨会带来什么恶果，也可以重申你们的使命并证明哪些地方正进展良好。我不主张当场与你的团队成员争论他提出的具体问题。你应和他在1对1的会议上合力解决这个问题，而不是当着其他团队成员的面争论它。

10.3.4 使用鱼骨图等工具来解决问题

在团队会议中，当试图解决一个问题时，你能做的最有效的事情是问“五轮为什么”。这是戴明提倡的方法，他建议你应该持续地问“为什么”直到找到根本原因，这个过程通常需要问5轮。一些团队试图直接头脑风暴出通用的解决方案，但这并非最有效的方法，戴明也不这么做。他喜欢通过鱼骨图来引出五轮为什么的答案。

例如，假设你有一个销售上的问题，把这个问题当做鱼脊（见图10-1）。



图10-1 鱼骨图的鱼脊

下一步，团队头脑风暴出“为什么我们销量少”的答案。不要将团队的想法在白板上写成一列，换种方法，把每个想法放在鱼脊身上扩展出来的线上，如图10-2所示。

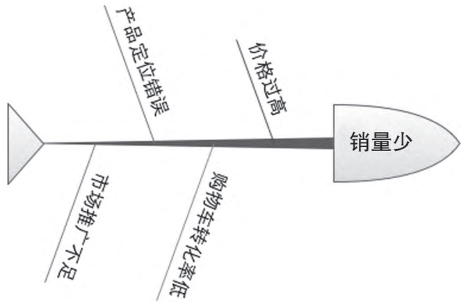


图 10-2 鱼骨图 - 第一轮为什么

添加完第一层鱼骨后，你再针对每根鱼骨继续问“为什么”。你的团队的产品专家应该知道市场推广和价格或产品选择是否改变。假设两者均未改变，那么是时候问问为什么购物车页面用户没有转化了。将问题的答案添加到各自的骨头上去（图 10-3）。

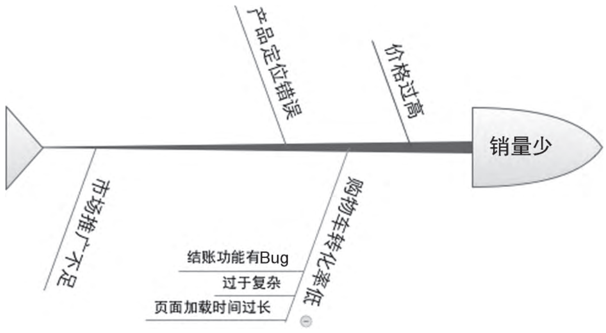


图 10-3 鱼骨图 - 第二轮为什么

你和团队一起已经头脑风暴出一些原因了。你知道你并没有修改购买过程，但里面可能有 Bug。你的测试团队同意针对产品再做一轮测试。但你还注意到页面加载时间太长了——99% 的时间内都是 5 秒！为什么页面延迟会这么高？（见图 10-4。）

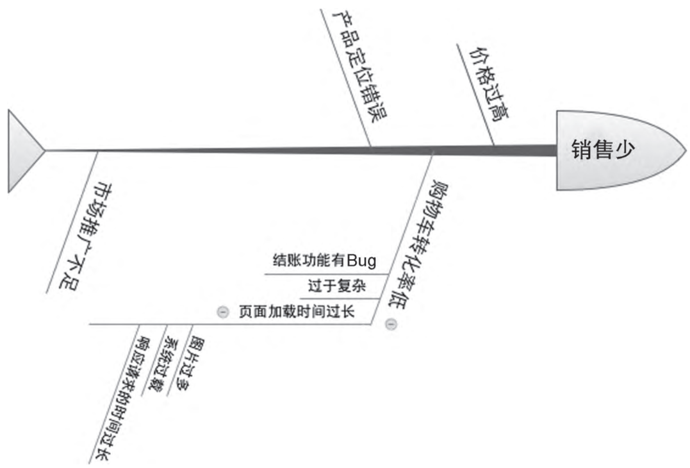


图 10-4　鱼骨图 – 第三轮为什么

让我们应用些数据到这个例子上去。系统正运行在 30% 的 CPU 负载上，设计师也认为你的图片已经是最小且拼合[①] 的了(这是针对图 10-4 中可能的原因的回答)。那么原因必须与请求的响应时间有关了，事实上这个响应时间在99% 的时间内是4.5 秒！为什么延迟这么高？（见图 10-5。）

这个问题渐渐变得有趣了，鱼骨图也渐渐复杂了。但你还未找到根本原因。根据第 9 章的信息，你应该派个人去调查下是否有服务链问题，但会话服务器并没有引发任何问题。最后只剩下支持数据库了，它比你的预期要慢，为什么？（见图 10-6。）

啊哈！假设你的团队没有足够的专业知识来搞明白为什么支持数据库会这么慢，那么合理的下一步是会后进行调查，咨询相关的数据库架构师。这个数据库架构师仔细看了数据库后，发现你有相当高比例的缓存缺失，这引发了大量的磁盘读取（见 9.3 节更多关于缓存的信息），这就是这个问题的根本原因。幸运的是，当他调整了分配给缓存的内存容量后，你的产品销量上升了 10%，不出意料你也获得了晋升。干得不错！

①指使用了 CSS Sprite 技术，这种技术允许你将一个页面中所有与样式有关的图片拼合到一张大图上去，以减少图片请求数。——译者注

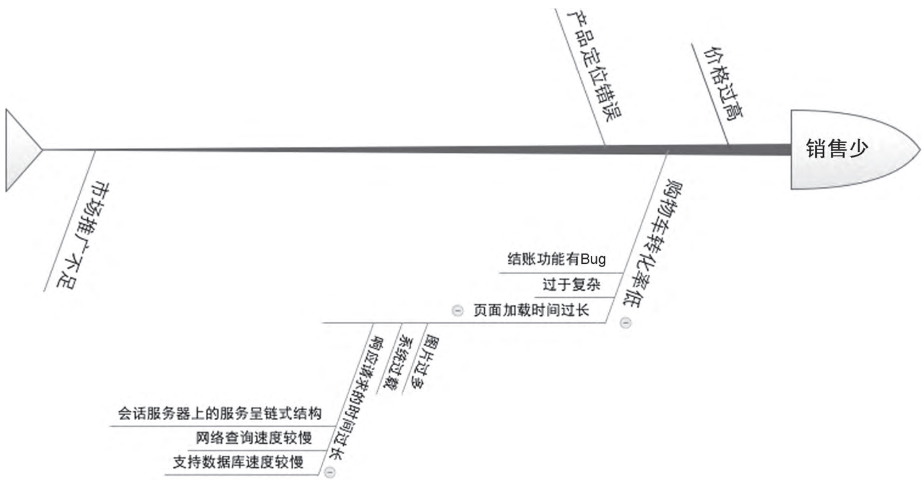


图 10-5　鱼骨图 – 第四轮为什么

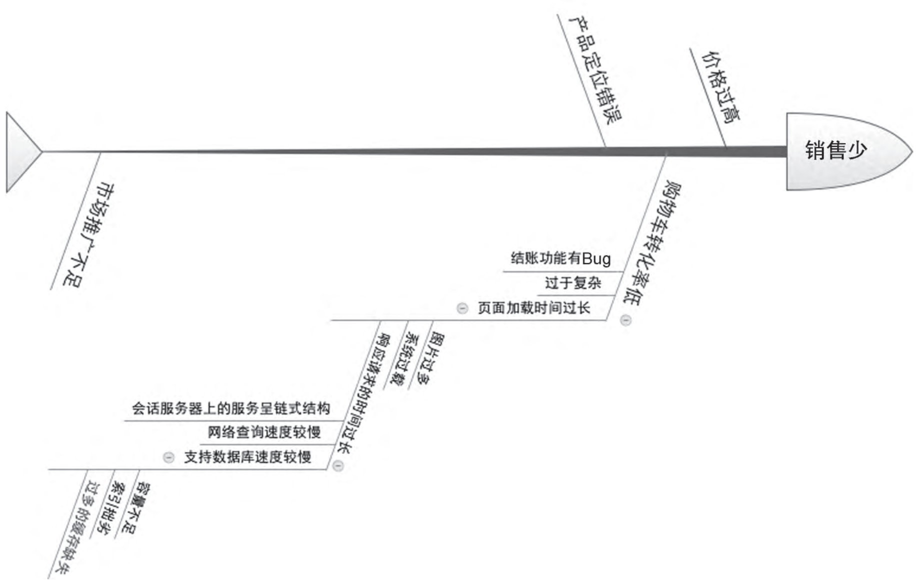


图 10-6　鱼骨图 – 第五轮为什么

你也可以在不使用鱼骨图的情况下应用“五轮为什么”方法。首先写下每个潜在的问题，然后反复讨论这些问题，最后将讨论结果写在纸上。这个过程有助于团队成员意识到他们已经想清楚了这个问题，你也会对自己是否解决了正确的问题更有信心。你可能需要多个会议或几周时间来完成这个过程，但只要能找出根本原因，这些都是值得的。

在上面的讨论中，你一定要懂得自己要扮演的角色是什么。我们的目的是要找到非常专业的答案。然而你在这个对话过程中扮演的角色却不是找答案，而是促进讨论。你不停地在问“为什么”，你向那些懂得你所不了解的知识的专家求助，如工程团队，设计师，最后是数据库架构师。在调查之初你并不知道答案是什么，但你不停地在问为什么。非常棒！这就是为什么你能交付卓越产品的原因。

10.4　如何做好演示

如果正致力于交付卓越的软件，你会需要做大量的演示，比如通过演示来获取资金、提供产品更新信息，甚至是通过推销性质的演示来说服人们与你的团队合作。显然，制作出色的演示稿并出色地演示是一个非常重要的技能，但它需要花费数年才能精通。史蒂夫・乔布斯级别的演示是你学习的参照，但它需要海量的准备时间，而且你也不一定有那么多精彩的内容，所以通常大多数人都达不到那个标准。好消息是，如果你花大量时间去上关于演示的课、学习演讲、接受训练等，你会发现有一些基本的东西可以帮助你的演示取得成功。更好的消息是，我在下面写下了这些基本规则和技巧。坏消息呢？坏消息是演示还得你亲自上马。下面是总结出来的技巧：

- 将演示时间控制在 15 分钟内
- 永远传达且只传达一个信息
- 讲故事
- 制作“综述单页”
- 重点演示用户体验
- 极度专注倾听

10.4.1 将演示时间控制在 15 分钟内

你演示时间应该是 10~15 分钟。很多领导者习惯性地破坏这条关于演示时间的规则，他们非常深入细节，认为有太多数据要分享，并认为自己处理的问题极其复杂。这些都没错，只是他们忽视了现实情况。现实情况是人们通常得在 30 分钟内做完所有事情，包括传递信息，讨论它以确保受众充分理解，并最后得到批准。即便你幸运地被安排了一个小时或更长时间，也一定要记住对于一贯忙碌的高管来说，每个会议他们只有 15 分钟会专注于你所讲的内容。

在 30 分钟内完成一次陈述时间为 15 分钟的演示一般遵循下面的时间线：

00:00 ～ 00:05（5 分钟）

等待。每个会议似乎都会延迟 5 分钟开始。

00:05 ～ 00:15（10 分钟）

进行演示

00:15 ～ 00:25（10 分钟）

针对你的演示进行讨论和答疑。在演示过程中进行也是可以的，这样演示会耗费更长的时间，但产生的问题会减少。切记你必须在 20 分钟内完成演示和答疑。

00:25 ～ 00:30 (5 分钟)

重述结论和重要反馈，并就后续计划达成共识。如果听得非常认真，你或许可以在 5 分钟内重述它们，这样会议可以更早结束，你的听众也会点头称赞。

从这个时间线可以清晰发现，你需要尽量将演示时间控制在 10 分钟内，且绝对不能超过 15 分钟。如果你所在公司的会议通常都准时开始并准时结束，也许你可以将演示时间拉长 5 分钟，但我建议你不要这么做。你永远不知道什么时候一个关键的利益相关者会被叫到走廊上谈事情或不得不接一个来自托儿所的电话。最好假定每个会议你都会失去最少 5 分钟。这样你需要将你的内容压缩到 5 分钟，这看起来似乎不太可能，但只要你始终只传达一个信息就能做到这一点，这就是本书下一条要讲的东西。

10.4.2 永远只传达一个信息

无论你的信息是“我们有一个杀手级的创意需要资金”、“我们必须决定是以消

费者为导向还是以业务为导向”，还是“按照目前进度我们有 90% 的把握赶上发布日期”，都得有一个关键信息。如果你连关键信息是什么都不知道，还是取消演示吧。

你或许认为你有两个关键的信息都需要传递，但最好先只传递一个信息，然后再约一次会来传递另外一个信息。有三大理由证明为什么你应该每次会议只传递一个信息。

- 第一，试图传递两个信息可能导致彼此混淆。你的听众也许很聪明，但他们刚从一个毫不相干的会议出来，接下来还要去另外一个毫不相干的会议，所以他们一时半会难以完成场景的转变。如果你能让听众们轻松一些，他们会喜欢你的。
- 第二，在讨论过程中你的脑海中会有两个想法左右互搏。一方面你会下意识地让听众们分清你带来的两个议题的主次关系，一方面你又希望他们能专注于你的内容，而不是排定主次。兼顾两个信息的传递还会让你难以掌控会议。
- 第三，也可能是最重要的理由，你需要将演示时间控制在 15 分钟内，不过因为你是这方面的专家，你可以将时间控制在 10 分钟。而在 10 分钟内传递两个信息的可能性在 0 到 -1 之间，简而言之，你没有时间传递两个信息。

一旦准备好了你要传递的信息后，始终围绕它来做演示。去掉与这个信息无关的数据或议题。如果你还想保留，那么将这些支持项，如图表、客户声音等，都放到附录中去。要是讨论中涉及这些内容你可以再把它们翻出来。每页幻灯片的标题都应该建立在你需要传递的信息上。比如要做汇报近况的演示，你要传递的信息是“一切都在顺利进行中”，那么放置 Bug 燃尽图的那页幻灯片的标题可以是“发现 / 解决率正趋向 2/5”。你可以在这句话后面再加上“这证明了……”。所以，这页幻灯片的标题可以是“发现 / 解决率正趋向 2/5——一切都在顺利进行中”。如果你演示中的每页幻灯片都围绕着关键信息，即便听众此时正在查收邮件，也不用担心信息传递不成功。

制作完演示稿后不要忘记多检查几次，务必确保它就是你要传递的信息。随着思考的深入，你会经常改写一些文案以便更好传递信息。这非常好，在演示稿制作阶段我们恰恰希望能有这种思考，但记得回头仔细检查，确保它仍然专注于传递那个你唯一要传递的信息。

10.4.3　讲故事

人们喜欢听故事。故事有代入感，能将你要传递的信息与真实的生活连接起来。不管在什么场合，只要你试着用讲故事来配合演示，就能取得更好的效果。让我们举例证明。比如说你有个杀手级的创意需要资金。在你的演示中，你可能像大多数人一样，一开始就讲创意细节，如：

通过将社交分享移植到移动端，使用自动生成的元数据和UGC，我们就能够构建出一个从根本上提升参与度并能病毒式增加安装量的应用程序。

是的，这是一个差劲的、糟糕的、没有任何亮点的烂到极点的说辞。里面有缩写词、有故弄玄虚的字眼，还讲得很含糊。如果使用双倍的语言并以故事的方式来讲述又会怎样呢？

想象一下你正在酒店里品尝着人生中从未吃到过的相当美味的龙虾，而且不止你是这样，你的妻子也是第一次吃到如此美味的龙虾。这是一次美妙的体验，等到回家后，你把这些都写到了Facebook中，但你累了，嫌烦了，自然记录得不怎么完整。但如果你能在智能手机上通过点击一个按钮来分享这个体验会怎么样呢？我们能做到它：通过一次按钮的点击，程序会自动带上酒店的名称、你的名字、你妻子的名字和一张照片。

现在开始演示就很合适了。不妨展示一张你满脸笑容的妻子吃着龙虾的照片并就着它开始讲故事。你甚至可以将这顿龙虾晚餐的费用作为开支申请报销。你就这样讲述了一个赶上乔布斯水准的故事，并且讲述所花的时间比听众理解你初始说辞的第一句话所需的时间还少。除了去掉了故弄玄虚的字眼和缩写词，你还通过讲故事做到了其他五件事情。

- **将创意与个人生活关联起来，从而使听众产生了代入感**。在酒店吃晚餐这件事大多数人都能理解。
- **让观众跟着你的节奏走**。使用“想象……”和“如果……会怎样呢”这类话让你成功地吸引听众短暂地将注意力集中到你身上，然后你再通过故事本身激起他们的兴趣。
- **提供了一个人们都能理解的具体例子**。新想法都是不好理解的，观众需要一个参照系以便赶上你的步伐。在这个例子中，出去吃晚餐和想用智能手机分享都

是人们能够理解的东西。在电影行业，人们喜欢这样说一部片子："它就像《虎胆龙威》加《阿甘正传》加《黑客》，不过还是花了些心思。"之所以这样说是因为具体的例子给了忙碌的高管一种快捷了解新概念的途径。

- **描述了你将要解决的问题**。并且你描述得很清晰。
- **描述了你的解决方案将如何提升用户的生活品质**。你应该记得在第1章我就提到过谷歌的拉里·佩奇和亚马逊的杰夫·贝索斯都坚信应把用户放在第一位。你需要在故事中传达你将如何提升用户的生活品质，在这个例子中你做到了。

我经常看到偏技术的演示者不去描述用户场景，而是大谈特谈特性细节。有决断力的高管常常会在演示中间叫停演示并强制演示者清晰阐述用户场景，高管会这样说："停一停，我需要一个例子，比如以我妹妹莎朗举例。她使用的是一台智能手机，然后我们在外面吃晚饭。接下来会发生什么？"一旦发生这样的事情，你的风险就大了，你必须沿着高管设定的故事框架讲下去，但这并不容易。所以你最好自己先讲故事。

遗憾的是，一些演示并不适合讲故事。说些像"让我们讲一个团队的故事，它是一个勇敢的团队，一个高尚的团队，一个需要推迟发布日期的团队……"这样的话可不是个好主意，所以要适可而止。在近况汇报会议上，你只需简单地列举证据来支持要传递的信息即可。其他类型的演示，如用户体验评审或推销，适合讲故事的原因是你能围绕用户组织内容。

10.4.4 制作"综述单页"

谷歌前工程总监彼得·威尔逊发明了"综述单页"法。彼得也许不是史蒂文·乔布斯式的演说家，但他非常善于传递要点并让注意力过于分散的高管转入正题。他将创业公司以"人才收购"的方式卖给 Facebook 也印证了他讲故事的能力。

彼得的"综述单页"法是指演示稿的第一张幻灯片，即位于标题页之后的第一页，必须包含你演示的关键要素。当管理层中有怀疑论者想要直接看结论时，你就翻到这一页。通过将整个演示内容压缩到单张幻灯片上，你可以打消他们先入为主的想法。

"综述单页"的另一个好处是当在演示过程中发生跑题现象时你可以翻到这页强调你的演示主题。投资人或高管纠结于你认为无关紧要的事情的现象还是很普遍的。不只是你，它也一次又一次地发生在其他人身上。如果准备了这张特别的幻灯片，你就能快速翻到这页并利用它来引导对话回到正轨。不幸的是，"综述单页"法和"讲

故事”法并不兼容，因此你需要谨慎使用这种方法，特别是你的高管喜欢快速往后翻的时候。

“综述单页”应该包含以下四块内容。

1. 你想讨论的东西是什么

避免缩写词和新名词。你几乎肯定想使用缩写词来节省空间，别这样做，你的听众可能知道也可能不知道这个缩写词，而你应首要考虑如何让听众免于思考。代号也可能使听众分心或困扰，你的老板或投资者不太可能听说过你最新的代号。需要向高管解释的东西越少越好。比如将第一点写为“结账体验糟透了”就是一个清爽的好例子。

2. 机会

换句话说，你和高管们愿意花费时间讨论这个议题的原因是什么？例如，“80%的用户在结账过程中放弃了，这代表着一个价值 1 千万美元的机会。”注意我将两个基本数据点放在了这里。你也可以将演示的核心数据压缩成 1 到 2 个重要数据点。将这些数据点添加到“综述单页”中能增加其合理性并在开始阶段就打消受众的疑虑。如果招聘团队主管的一个主要标准是“必须会用数据说话”（见第 8 章），那么在这张幻灯片中使用数字有助于老板确信自己雇佣你是正确的选择。

3. 提供的解决方案

继续上面的例子，你需要让解决方案具象化，但“综述单页”中并不适合使用图片。请用清爽的文字来阐述解决方案，如“增加未登录用户购买流程。将购买步骤从 5 步减少到 2 步”。添加原型图的链接，这样当需要时你可以快速将它拉出来。

4. 成本和实施时间表

某些情况下它还可能是你要问的内容，你需要管理层给你这些东西。它可能是“2 个工程师工作 2 个月”或“同意发布”。然而你不能到这结束了，如果你的管理层打算给你承诺，他们也需要你给他们一个类似的承诺。你需要提供一张你能负责的时间表。例如，你可能写“2 月份发布内部测试版，5 月份交付”。

图 10-7 展示了一张完整的“综述单页”是长什么样的。它缺乏视觉设计（我是故意的，因为不想让人们认为我华而不实），但你可以发现要了解我们要谈论什么、我们需要什么、当演示顺利通过时我们要何时交付是多么简单啊。

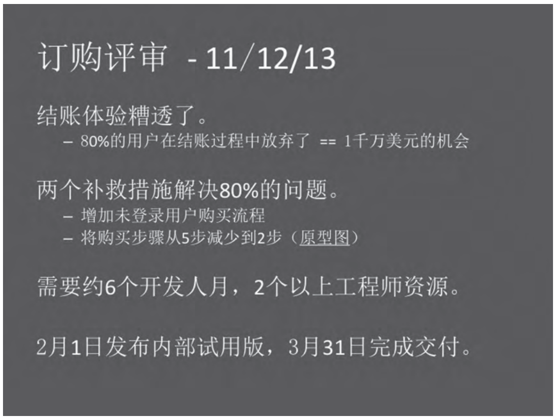


图 10-7 “综述单页”

一旦你写完了这张幻灯片，就可以准备演示稿的其他部分了，最终演示稿将包含四张幻灯片和一张专门说明行动计划的收尾幻灯片。五张幻灯片的演示稿可以较为宽松地在 15 分钟内演示完。

10.4.5 重点演示用户体验

打破成见并在脑海中建立新的具体图景的最佳方式之一是使用图片。具体来说便是重点演示用户体验的原型图。先展示几张当前用户使用网站的截图。我们通常将它们放在图 10-7 所示的“综述单页”之后的幻灯片上，以说明当前 5 步的结账路径是多么糟糕。这些图片有助于巩固听众脑海中对问题的认知。此外，当你向他们展示美观的未登录用户购买过程界面时，他们会因为强烈的对比而欣喜若狂。

在重点演示用户体验时仍然阐明关键信息。你可以使用“综述单页”法来阐明信息并为会议设定背景。或者你可以简单地在标题幻灯片中表达它，比如：“产品评审之 iPhone 端应用的用户体验评审。”在会议开始前的等待时间里听众可以通过看标题来清楚知道要求他们做什么。在这个例子中，作为产品评审会的与会人，他们需要评审并决定是否通过 iPhone 客户端应用的用户体验。

为了从外部用户的角度来描述你的原型，请首先说明首要用户目标而非描述特性。然后你可以指出用户体验是如何满足这个目标的。如果产品用户体验设计得好，那么首要用户目标便是用户界面中最易被发现以及最简单的那部分。你可以将用户界

面分成若干张幻灯片，每切一张幻灯片就好像用户点按一下。这个方法可以让听众把自己当做用户来持续思考这些视觉呈现，并有助于构建一个强大的叙事场景。

10.4.6 极度专注倾听

如果到目前为止你都是按流程走下来的，那么只剩最后一个不可或缺的工作了——倾听。虽然一边倾听一边演示是相当困难的，但你没有别的选择。你一定要注意分辨反对意见的细微差别，看看哪些是不可通融的，哪些又只是抱怨。能否分清建议和需求决定了你是能卓越地完成交付还是什么也交付不了。

一个有效地捕捉各种意见细微之处的方法是让一个值得信任的同事一字不差地把它们记下来。精确记录现场说过的每一个字能减少主观争论的空间。你们仍然会就每个字的意思进行争论，但不会去争论谁到底说了什么。

如果没有人能帮你一字不差地把别人说的话记录下来，你就应该自己带个本子把那些最重要的利益相关者的反馈记下来。提前了解谁是关键的利益相关者将使你聚焦在最重要的反馈上，并让你高效地分配记笔记的时间。比如在谷歌做演示时，我懂得每次都应关注埃里克、拉里、谢尔盖说的话（就是这样的顺序），其他高管的指导可以作为有益补充。不过最好的建议常常来自这些高管，但他们不能阻止你的产品发布，所以他们的意见即使再好也没那么关键。同样的方法也适用于杰夫·贝索斯和他的高管团队（简称“S 团队”），逐字记下杰夫的话很关键，然后把高管团队的指导作为有益补充。

在演示最后，如果你对某些重要利益相关者说的话尚不理解，必须趁现在问清楚。比如杰夫之前有提一个建议：“你看可不可以这样做……”然后你需要重提这个话题：“杰夫，我想把这点再确认下——你觉得我们应该推迟发布去做 X 吗？”尽量少做这种事，除非必须要这样做。不要过度推销。曾经有次埃里克·施密特打断我说：“不用继续说了，你通过了，下一个。”如果你用计划一半的时间就开完了会议，赞！

10.4.7 更多的演示技巧

- 如果一页幻灯片上不只是一幅图加一句话时，请把这页的信息要点放在标题里。苹果那帮人很擅长这个技巧，不过在户外或对较大规模人群进行演示时这样做的确是最好的。

■ 如果你发现先前的技巧并不适用，请把信息要点放在标题里。

■ 如果你没有模板，请使用基本的商学院的那种蓝色背景，配上黄色和白色的字。它们在任何投影仪中都能得到很好的呈现。

■ 阅读并遵循罗宾·威廉姆斯所著的《写给大家看的设计书》一书[①]中提及的基本视觉设计原则。

■ 阅读并遵循爱德华·塔夫特所著的 *The Visual Display of Quantitative Information*, *2nd Edition* 一书中提及的信息展示原则。

■ 不要担心留白。人们喜欢留白多的幻灯片。

■ 不要使用“构建版本”这类字眼，除非你想让听众认为你华而不实。

■ 不要使用红色来代表除危险或糟糕之外的任何其他意思。

① 中文版有精装版和平装版，全彩印刷，人民邮电出版社出版。——编者注

第 11 章 胜在决策

11

SHIPPING GREATNESS

软件和英语的区别并不在于单词，C++ 使用的很多单词狄更斯也使用了。它们的区别在于软件是决策的物质载体。因为你可以用软件来做任何事情（不要让任何人告诉你有东西做不了），所以你们产品的骨架取决于团队的决策——你用它来做什么、怎么做。

不幸的是，做决策不是简简单单地说“行”还是“不行”。软件不像大多数英文编撰物，它的复杂性决定了它是团队共建的产物，因此它也是团队决策的反映。有时候大老板可以独断专行，但不幸的是你并不是大老板，所以你必须设法让你的团队成员对他们所钟爱的事情说不。下面就是做这件事的方法。如果它有点像哄正在看《托马斯和他的朋友们》的小孩子上床睡觉的话，我只能说抱歉了。

首先你会先设法推后请求。“我们明天再继续看。”你会对小强尼说，他是蒸汽火车头托马斯的头号粉丝。如果强尼开始大哭大闹，而你又容易上当，你会试着进行谈判：“行，行，别哭了……再多看 10 分钟行了吧？ 10 分钟噢？”谈判过程可能很棘手，强尼肯定闹着要 15 分钟，但如果学习了本章内容，你也许能找到自己和强尼都满意的折中时间。当然强尼也可能一味地折腾和对抗，出现这种情况时，你需要祭出所有关于冲突管理的法宝来应对，如理解绝大多数冲突都是沟通不畅造成的，了解是什么导致了强尼这种反应，使用角色模型来使交流变得客观。我不确定角色模型是否能帮你应对强尼，但它们能帮你在商业谈判中让冲突回归客观。让我们逐个了解这些方法的更多细节。

11.1 推后：“我们明天再完成。”

由于人类畏惧冲突的天性，很多软件团队主管害怕说不，于是“蠕变特性”（Featuritis 或者 Creeping Featurism）① 成为一种常见病。有时候人们也不是害怕产生冲突，而是害怕做得不够好。对于软件团队主管来说，他们常常害怕软件的功能还不够多，而他们又最知道软件有多少功能。于是这种害怕催生出来的设计会使产品过于复

①指产品由于不断增加的特性而渐渐臃肿。——译者注

杂而无法发布。下面有一个简单方案可以解决该问题。

将不属于绝对最小化可行特性集的所有特性放到 V2。检测这种特性的方法是："这样的软件做出来后用户能完成他们的基本任务吗？"如果用户不需要这个特性也能完成任务，那么即便任务完成得再糟糕、再丑陋，你也可以把这个特性放到 V2。你必须坚持进行检测，因为每行代码（除了单元测试）都会降低你发布的可能性，而没有发布，卓越发布就无从谈起。

如果你在建议将某个特性放到 V2 时遇到阻力，那么是时候谈判了。

11.2 谈判："行，再给你 10 分钟。"

几乎所有特性或用户体验的争论最后就会变得像绑匪谈判。你控制了人家的孩子，或者人家控制了你的孩子，除非某一方开出的条件被接受，否则孩子就会被撕票。无论你们是争论某个 Bug 是否是阻碍者，还是争论图标该是绿色还是蓝色，这都是谈判——而处在危险中的孩子就是你的产品。如果你是大老板，可以抱起孩子就跑，可惜你不是。你也许基本上没有或是没有正当的权力，因此依靠谈判快速达成共识是通往成功的必经之路。

谈判的第一步是正确地认识到不管你是不是产品的负责人，你都不是老板。你的团队之所以和你一起工作是因为他们喜欢你或者你的产品，而不是因为别无选择。软件行业中的一个现实情况是：如果你想要与某人合作，别人也都想要与之合作。因此你一定要让你的团队成员参与决策，让他们与你共同拥有这个产品。

团队主管常常会召开小范围的经营决策会议，这是错误的。它表面上看似乎很有成效，但将你的工程团队排除在会议之外，就等于将他们排除在决策制定过程之外，而将他们排除在决策之外会导致他们从产品开发过程中抽身而去。因此要想交付卓越的产品，你需要让团队成员体验到参与感和发言权。

更好的团队决策制定过程是在早期就让所有团队成员都参与进来。根据我的经验，这种方式从长期看来更富成效，因为你有机会在项目早期就向整个团队解释商业目标和团队目标。如果换成和级别较高的利益相关人开很多小规模的会议然后直接向团队传达制定好的完整计划，你未来可能需要更多的小规模会来处理后续出现的各种忧虑。更糟糕的是，有些忧虑是合理的，你将不得不大量修改计划，而优秀的项目管

理方法告诉你应尽可能在开发过程早期完成所有修改。

管理大师彼得·德鲁克在他的《卓有成效的管理者》一书中对此有些不同的看法，他主张你需要组织拥有清晰目标和相对较少参与者的会议。虽然他的建议和我有点不同，但他也认为卓有成效的管理者应该将会议的议事日程公布给所有相关人员，并计划在会后发出清晰的会议纪要。这样相关人员参与会议，其他人员也能知道前因后果。因此他的方法和我的建议在内容层面是基本一致的，你大可放心地听从彼得的建议，尽管它没那么讲究协作。

如果在你思考让谁参与决策时有什么指导性原则需要遵从的话，那一定是信息透明。请将决策的理由透明，将决策的时间透明，将团队参与决策的途径透明。

一旦将关键人物聚到一起后，你需要通过谈判而不是击败别人来达成共识。很多软件主管都有点大男子主义，这种性格导致他们首先考虑如何击败对方。在谷歌有一个经典案例：一个年轻气盛的产品经理去参加命名评审会以决定他产品的对外名称。没过一会儿他就给市场团队留下了深刻印象，他说："就让我们叫它 Google Turbo 吧！"然后惨遭嘲笑。大男子主义就是这样，爱发号施令，咄咄逼人，以及偶尔还会滑稽可笑。

因此我丝毫不惊讶很多好的软件主管是女性，尽管主要是男性从事该职业。安妮塔·伍利和托马斯·马龙对此进行了一些研究，为我们提供了出现这种情况的原因的深入见解[①]。研究者们在比较了个体贡献者的IQ和由他们组成的小组的IQ后发现了一个不同寻常的强相关性：小组中女性的比例越高，小组的集体 IQ 就越高。在他们看来这并非是因为少了一个 Y 染色体。他们认为在研究中这些女性将更多的协作技能带入测试中，而那些"大男子主义"类型的人则倾向于像老板一样决定答案。

我非常认同伍利和马龙的结论。Google Apps 和 Google+ 的重要贡献者、工程经理布莱恩·马什曾说团队主管"需要学习如何在团队规模增加的情况下等比例提升团队的前进速度"。上述研究中团队里的女性比单干的女性更为能干，她们能够促进共识的产生，而男性则不行。换句话说，靠她们的协作方法能产生一个比 Google Turbo 要好得多的产品名称。

再换种方式来理解这个事情。在老一套的妥协中，你和你的团队成员各赢 50%，

---

①http://hbr.org/2011/06/defend-your-research-what-makes-a-team-smarter-more-women/ar/1

也各输 50%。但现在还多了一种选择，如果你和团队成员能共同努力去创造性地达成共识以完成你们的集体目标，你就赢得了 100%，从而得到了研究中受女性影响的小组得到的东西，一个比各自单干要明智得多的结果。

因此你的目标是促使产生一个满足各方需求的创造性的解决方案。哈佛谈判项目（The Harvard Negotiation Project）认为这个过程的首要关键步骤是达成对目标的共识，该项目因罗杰·费希尔和威廉·尤里合著的《谈判力》一书而广为人知。

假设我想在我的 Hello World 应用程序中使用你的通讯录服务。我礼貌地发出请求，你起先拒绝了，因为你已经处于不充分提供的状态且你的服务器也没有额外的容量。我打赌这种情况很常见，对吗？

在这个冲突中，你我可以表述以下目标并达成共识。

- 你不想让你的服务器被我的请求拖垮。
- 你已有的客户端不能因为我的使用而导致服务品质下降。
- 我需要使用你的通讯录服务来提供更好的服务品质。
- 我们都不想变成蠢货。

这些目标看起来很合理，我们也认为它们很合理，所以我们首先就这一点达成了共识。然后可以据此创建很多满足所有条件的解决方案，如将通讯录数据本地缓存在我的服务器上，给你的集群增加更多容量，或者内建支持返回 HTTP503 错误且响应域包含指向数据库只读版本的链接等。拥有创建满足各方需求的共赢的解决方案的能力是作为领导者最重要也最令人满意的方面之一，它还会鼓励你将每次谈判都视为创建解决方案的机会。

有时候甚至还未讨论目标你就陷入谈判中了。这时候有三种帮助你回归正轨的方法。

- **聚焦于促进**。不要一开始就尝试去解决问题，否则你会带着先入为主的观点以至于无法保持中立，这会使讨论更加复杂。你应该首选确保听到每个人说的话。重点关注那些外向的人，他们常常会说很多。内向的人不太喜欢在人群中发言，但当他们发言时你务必认真倾听。
- **“先寻求理解，再寻求被理解。”**个人成长导师史蒂芬·柯维在他的《高效能人士的七个习惯》一书中提出了这条原则。它说得非常到位。我发现一个组织中最具权势的那些人也是最不擅长交流并处在极不正常压力下的那些人。因此，你需要极其努力地理解他们到底在说什么。问问你自己，他们真正关心的是什

么？然后通过提问来验证你的假设。

佛罗里达州立大学电影艺术学院院长弗兰克·帕特森有一次教我了一个和他那儿演员合作的范例。在试着为一个演员指明方向之前，先清晰表达你所观察到的东西：“我听到你在说……”通过将信息反述给信息的发送者，你使得那个人可以纠正你的错误以减少误传。这个方法很出色，它强调了你渴望先寻求理解，再寻求被理解。

- **如果你已经有了倾向性的判断，不妨先说出来，然后让其他人继续发表观点。**

  我认为当其他人已经知道你有倾向时，你不妨开场先阐明你的倾向，然后把指挥棒交给其他人以便能专心倾听。这个方法真诚且高效。

到现在为止，我们一直聚焦在谈判和协作的通用技能上，但有一个常见的场景需要一些稍微专业的技能：财务谈判。你早晚都要面对真实金钱的交易，而不仅是在大富翁游戏中玩过家家。你会对第一次甚至第 1000 次交易都满怀畏惧，尤其在你拥有这个公司并且花的是你的钱的时候。然而几乎所有的交易都很简单，即便它们感觉有多疯狂。我做过好几次这样的交易，包括在谷歌的两次企业收购，我认识到财务交易的谈判就像悲伤的情绪：它们分好些阶段，要是你了解它们，你就能更好地应对你的生活和交易结果。

11.2.1　阶段 1：这不是你的错

财务谈判几乎总是会令人心烦，究其原因是因为商业社会里公司媒介精英为了实现自我价值而将谈判的强度拉得越来越大。如果你认为你生而为人的价值仅仅是可以从带宽提供商那边获得略微高点的定期服务费折扣，我真心为你的家庭感到遗憾。请寻求心理咨询后再往下读吧。

如果你看到谈判方那边有这种症状的人，我只能再次表示遗憾。立即退出谈判，或者如果有一个理想的中间值，接受它，不过你还没有解脱。如果还想继续深入，你还得满足这个大男子主义的讨论鬼更多合法但很不合理的请求。接受它吧，不要让它影响你的生活。我发现有个方法很能调剂心情：提醒自己他也许正过着不幸的婚姻生活。

如果你不相信我说的财务交易是多么大男子主义且多么受媒介影响，你可以参考下面一些交易过程中的典型用语。

- “该把和服拉得更开一点了。”

■“我的已经摆在这里了，现在看你的诚意了。”

■“想要一起参加舞会？”

■“跳舞解决不了问题，最终我们得回到正事上来。”

呸！就当是帮帮你自己吧，不要使用这种无下限的攻击方法，不要使用这些脏话。如果你发现有人在使用它们，就让它们来提醒你他正处在阶段 1 吧。要求这些人停止污言秽语并进入阶段 2。

11.2.2 阶段 2：公平谈判并使用数据支持

既然不再抱有“我必须像戈登·盖柯[①]一样争取尽可能低的价格”这样的态度，你就可以公平地开始谈判。最公平地谈成某个数字的方法是参考交易数据。例如，你主动提供一些数据：“我能从 AT&T 以每兆 1 美元的价格购买带宽。”然后对方也主动提供额外的数据：“我们的价钱是每兆 0.95 美元。”

理想情况下到这个点你们就能谈成了，最终价格将定在 0.98 美元或 0.97 美元（取决于谁更流氓一点），你们达到了共赢。你的服务商得到了微小的赢利，你也得到了微小的折扣。

要是大多数谈判都是这样该多好！不幸的是，你可能会说：“我只愿意付 0.75 美元。”他们则可能回应道：“我们需要收你 1.25 美元。”这时候你们无法达成共识，只能进入阶段 3。

11.2.3 阶段 3：那个数据没谈成……我们编个新数据吧！

由于你或你的对手会不断披露和编造新的信息，因此财务谈判会耗费很长的时间。编造信息特别有挑战性。你可能这样编造你的构建成本：“这会花费我 5 个工程师 1 年的时间去构建，所以我最多只支付 100 万美元。”

一旦你越过这条线（你肯定会越过这条线），你的对手也会开始编造信息：“是的，但你能提前 6 个月进入市场，这至少值 500 万美元，所以你至少也应该愿意支付 400 万美元。”

这个阶段终究会过去，但在此期间你就像在参加一场拍脑袋式的军备竞赛，一

---

①电影《华尔街》中臭名昭著的金融家。——译者注

直在拼命寻找最适合登陆的中间地带。接受必须经历这个阶段的事实并设法快速通过吧。

11.2.4　阶段 4：寻找可免费提供的东西

最终双方都会对争论需要多少工程师花多少个月来整合，每个客户到底值多少钱，微软、谷歌或苹果构建类似的技术需要多久等问题深恶痛绝。在阶段 4，各方都会设法往交易中塞一些没有价值、无须成本的扯淡的东西。“我们有重要的发布计划，”你会说，“我们可以让你出现在莫斯科尼会展中心的讲台上。”对方甚至懒得对你这个市场策略予以置评，其实你也不应该期待对方能给什么反应，你只是希望若能给出更多“好处”，他们就会认同你在阶段 3 提出的最后数字。

这个“诱人的交易”阶段从很多方面来看都是各方在现实之下所寻求的一种心理安慰，大家都不想让对方觉得自己像个土匪，那样多不好。不过这点我们总是无法得偿所望，尽管土匪不是好东西，只有电影中的“企业狙击手”才听起来有魅力（见阶段 1）。

11.2.5　阶段 5：转身离开并安静思考

有可能在经历了几个月的谈判并额外奉送了一些好处后，所有人都疲倦了以至于愿意达成交易。这时你只管去做吧，直接进入阶段 6 并在回家的路上去庙里上根香，别忘了再给点香火钱。

但我们大部分人都没那么走运，因为疲劳会让所有事情更糟糕（大概是初为人母的朋友们告诉我的）。很可能阶段 1（你想成为戈登·盖柯——宇宙之王的那个阶段）又会抬起它丑陋的头。装腔作势开始出现：“行，我们的分歧太大了，我猜我们不得不各玩各的了。”威胁开始出现：“我们会用任何方法把你搞破产……”电话挂断后带有严重攻击性的脏话也脱口而出。

这时候一个必要的缓和情绪的时间自然就出现了。其中一方会离开谈判场所，或者因为苦恼而不回电话，总之不管怎么样，谈判暂时中止了。这对谈判来说是个好事，双方都可暂时离开并回到产品上面。这个重新拉开的距离允许你再次思考现实的情况。

产品是双方交易的基石。你需要它们，它们也需要你。如果你真的不能接受对方

在商业立场上提出的条款，没问题，转身离去即可。如果你能忍受这些条款，那么就咬牙接受它们。给谈判 2~3 周冷却的时间，再重新从阶段 2 开始。不过这次有些不同，你会愿意分享更多数据并支付更多金钱。对方也会愿意分享更多数据并降低报价，因此你们将更有可能谈成一个双方都能接受的数字。

11.2.6　阶段 6：协议、文书工作、以及互相指责

就在你觉得交易已经完成时，你会发现任何公平的、有价值的交易都还有大量的文书工作要做。但文书工作不是问题，问题是文书工作中一些细枝末节的东西会把你带回阶段 5 并使你不得不谋划一些针对该交易的“焦土策略”。这些细枝末节的东西之所以存在是因为任何合同都需要非常清晰才能具备效力。为了增加明晰程度，你必须定义一切细节，这时你会发现你们的沟通尚有很多不明之处，而这些不明之处会导致冲突。

如果双方都人困马乏且大男子主义，交易就会在这个点上滑出轨道，就像 25 美分硬币神话般地让美国国家铁路客运公司的火车脱轨一样。提醒你自己这只不过是 25 美分，不是一堆整整 1000 万的钞票，所以如果可能就随它去吧。请使用本章稍后提到的技巧来处理冲突，然后拼尽全力将火车拉回正轨。毕竟你已经同意和这帮人一起“参加舞会”，所以大度一些，去他们想去的饭馆、吃他们想吃的菜吧。

几个月后，你和你交易的团队很可能不再关心你付了多少钱了。几杯酒后，你估计只记得交易花了多长的时间。你可能回想一下说：“见鬼！怎么花了这么长的时间！”答案是大多数的时间都花在了阶段 1、3、4、5、6。是的，这些阶段都不是建立在确实的数据之上的。所以如果说这个故事有寓意的话，那么寓意就是*不要大男子主义*。

11.3　处理冲突

第三个也是最后一个你需要借以达成卓越共识的工具是冲突管理。我们从一开始就可断言一定会有大量的讨厌鬼存在，他们会制造冲突。你甚至可以认为和你一起工作的是一群独一无二的讨厌鬼、白痴和只顾自己的傻瓜。不过由于你是在软件行业，所以出现这种情况的可能性相当之小。更可能的是和你一起工作的都是拥有超低沟通能力和极高专业能力的人们。

“但是等等，”你可能说，“我和产品经理、工程副总裁以及设计师们一起工作，他们的沟通能力肯定不差！”所以他们必定是讨厌鬼（只是玩笑）。你可能是对的。更可能的是，这些人已经非常习惯和工程师以及你这样的家伙打交道，所以他们非常缺乏信心，担心你们会打乱他们的意图。这些人也会关心交付。也许他们关心的程度不如你，但他们的关心程度足以让他们害怕你会更改其设计、目标或质疑他们的决策以至于他们不得不进行调整。

管理专家托尼·施瓦茨曾写道，在这种情况下，你需要脑补出一个不一样的背景情况[①]。这里的背景情况是指你脑补出来的用以解释某些人奇怪行为的理由。但它终究只是一个臆想，一个建立在假设上的臆想。不要假设这场冲突的另外一方一定就是一个讨厌鬼，考虑另外一些可能的背景情况。

举个例子。刚加入谷歌地图团队时，我被设计团队搞得真心恼火。他们看起来聪明而友好，但我不断被他们气得想撞墙。当我问一些常规问题时，如“你要解决的用户问题是什么”，他们会回答说：“我们设计的目标是‘地图即用户界面’。你还是多花点时间去和其他产品经理聊吧。”

于是我倾向于认为地图的设计师们都是白痴，因为“地图即用户界面”不是一个设计目标，它对用户没有任何帮助。直到有次我一边喝着白葡萄酒一边吐槽时，我的一个设计师朋友给我讲述了一个不一样的背景情况。“他们没有业务目标，”他说，“就因为这个，他们被人随意抱怨了好多年，于是他们不得不凭借自己的力量来设法完成那些本该由你完成的工作。他们发明了一套组织原则来承受这些来自任意副总裁的任意微不足道的抱怨。而且他们可能把你当成又一个要抱怨的副总裁了。”

我知道不是天天都有朋友给你提供非凡的洞见，但你还是有办法来得到同样的结果并省下买葡萄酒的钱。你只要这样说：“的确我脑补出来的背景情况是他们都是白痴，但我实际没有 IQ 分数来支持这一点。也许真正的背景情况是上一个副总裁给了他们一个极不靠谱的目标，然后他们设法实现它，在这种情况下，他们捍卫这个目标是说得通的。”

如果你脑补出另一些可能的背景情况，你就能以更积极的心态来应对冲突。

等到你相信你所面对的家伙不一定是个讨厌鬼时，你就能找到问题的根源。90%

---

①http://hbr.org/2007/10/manage-your-energy-not-your-time/ar/1

的情况下冲突的根源都是沟通不畅。另外 10% 中有 1% 是因为你面对的的确是个讨厌鬼，剩下 9% 是因为目标的方向错了。

你真正需要了解的只是那 90% 的情况，你和对方在“各说各的”。换句话说，你们对重要的事情达成了共识但却纠缠在一些细节或措词问题上。我发现当工程师和工程经理真的关心某件较大的事情时，他们会经常把焦点放在细节上，如不要写重复的代码等，毕竟细节更容易被具象化。由于你知道这时你们有很大可能在各说各的，你便可以回到上一步并问问自己其他人真正想说的是什么。这是第一步。

一旦确定你们正在各说各的，你便可以借此机会建立一个通用的术语表，这样你们在讨论中就不会因为措词问题而导致沟通不畅。为了做这件事，你需要知道工程师和他们的团队经常会给同一个单词赋予不同的意思。在一个团队中“hack”可能是指一件可怕、危险的事情，在另一个团队“hack”却可能指一个不错的捷径。再比如我在谷歌地图团队时发现“地标”这个常见单词可能指的是一整类事物（酒店、旅馆和埃菲尔铁塔），而不单指埃菲尔铁塔这种事物，他们把埃菲尔铁塔这种事物单独称作“游览胜地”。这种概括和命名的惯例在软件行业合情合理，它就是软件的工作方式——以抽象的对象为基础并为这些对象添加属性以使之具体。因此，团队开发他们自己的专用语言是非常常见并完全合理的。

不过值得庆幸的是解决分类问题并不难。你可以说：“不好意思，为了确保我们在同一个频道上，我想问下你所说的‘地标’指的是什么？”工程团队将非常高兴地解释它们，因为他们喜欢定义东西。这也是他们从事编程的原因之一。

即使你们使用了同样的单词来代表同样的意思，也可能因为其中一方并不完全了解交流的背景而“各说各的”。与你共事的非工程师同事大多都在同时操心多个项目。他们并非一定要知道你为什么讨论这个话题，你也并非一定要知道为什么他们正变成一个蠢货。回到上一步，说：“我觉得我们可能在各说各的。”然后接收或者提供背景信息。例如：

为了确保我们在同一个频道上，我在这里先讲一点背景。我们正试着构建一个针对盲人用户的文本转语音的应用程序。不幸的是，我们想在消费电子展上发布，这意味着有一个硬性时限。而且我们的目标用户是盲人，所以我们不得不做些艰难的决策和优化。这样说你能理解吗？

这个例子披露了两个关键且常见的背景元素。第一，时间总是一个挑战，清晰

地说明时间上的限制有助于其他人了解什么是可能做到的。不相信时间会成为一个问题？让一个朋友打两个中等复杂的结，然后你和另一个朋友蒙着眼睛去解开这两个结，整个过程不准说话。再重新这项任务，不过要在 1 分钟内完成。你会深刻感受到这两次过程的不同之处，第二次你很可能遭遇一些令人心烦的、非语言的冲突，因此挑战虽然一样，但时限上的压力会让一切会变得更为糟糕。

第二，常常有一些别人并不知道的假设或者别人自己做出的假设。在这个例子中，设计的目标群体是盲人用户，这类用户的需求和一个典型的文本转语音的用户是不同的。当你给团队成员介绍背景时确保梳理出这些假设，因为它们常常会导致沟通不畅。

再补充一点，请设法确保你的项目名称在产品的整个生命周期中是一致的，否则你会让那些不知情的人们感到迷惑。如果你在一家小公司改名字可能无所谓，但在大公司就有问题。重新命名项目带来的收益很少能超过由此带来的沟通不畅所增加的成本。你还会因此而买更多的止头痛药埃克塞德林。

有时候别人听不懂你说的话，而你企图通过分类法和背景介绍来澄清它们也遭遇抵制或让人丧气时，请转向白板。因为这个原因，我总是把一面大的白板放在桌子旁。我发现图画和列表能使事物变得清晰。通过写下你所讲述的内容，别人就能把注意力集中在白板的图画或文字上，而不是随风逝去的音节。这个技巧虽然简单，但极其实用。

然而有时冲突并非由明显的沟通不畅或根本的观点分歧导致的。当有人戳中你不想被别人戳中的地方时冲突也会产生。你之所以知道你被戳中是因为你产生了打这个家伙一拳或逃离现场的冲动。这是一个经典的战斗或逃跑的反应。当有人戳中你时，请花整整一分钟的时间来想清楚刚刚发生了什么。然后停下来，说：“哼。”60 秒后你分泌出来的肾上腺素才会开始被身体吸收，20 分钟后才会被完全吸收。因此，不做任何事情并静止 60 秒将使你有时间避免从房间逃离或者采取报复行动。

如果想要更好地控制每个人都会有的强烈的情绪反应，你需要先识别出是什么引发了这样一个反应。是人们叫你孩童时期的小名来激怒你，还是工程师对你专业判断的质疑让你着实恼火？不管具体导火索是什么，请暗示自己当被戳中敏感点时，你将得到了一个控制自己反应的机会，你需要努力让自己的反应恢复原状。这样下去久而久之你就不会那么敏感了。

虽然你可以管理自己的情绪反应，但因为软件工作强度很高所以偶尔发发脾气是很正常的。大部分人们为自己的产品加班加点地工作，因而对于他们的心血结晶都格外维护。所以无须惊讶他们的情绪会很强烈，你最好马上接受这个现实：惹恼他们是不可避免的，尤其在你设法交付的时候。但如果你能客观地进行交流，也许可以减少惹恼他们的次数。

客观地交流是指只围绕软件、用户或问题进行讨论，而不把人牵扯进来。日常你可以使用一些有效的方法来帮助确保你交流的客观性并增加拿到一个没有压力的结果的可能性。我最钟爱的三个客观化的方法是：

- 不说“你”或“我”。
- 聚焦在角色模型上，而不是人上。
- 使用客观指标。

11.3.1 不说“你”或“我”

试着修改邮件中所有使用了“我”或“你”的句子。这样能让邮件内容变得客观而且聚焦在产品的表达上，而不是聚焦在“你”觉得怎样或“我”提出什么上。如果你在口头交流中也能去掉“你”和“我”，那就更好了。你可能会发现你会更频繁地使用被动语态，但相比起降低你团队成员的压力水平并提升他们的幸福指数而言，这点成本微不足道。谷歌非常推崇这个方法，以至于晋升委员会的成员都不能说“我觉得……”之类的话，而是应该说“晋升材料说明了……”。与之类似，议会礼节也不允许政客直接跟另一个人叫板，这样增加了一层缓冲并有助于让辩论不那么针对个人。

11.3.2 聚焦在角色模型上，而不是人上

另一个很有效的讨论问题的方式是使用角色模型，它就像你或许玩过的袜子玩偶游戏一样。袜子玩偶之所以经久不衰，是因为它们能让富有挑战性的交流变得客观，而角色模型也能做同样的事情。例如，使用角色模型之前你会这样说：“你想要的真不是先注册，你想要的只是看看产品是否还有货……”使用之后你可能说：“莎拉是一个顾客。她想随便看看然后决定该买什么。那么她想要做什么呢？”

引入莎拉这个角色可以大幅增加交流的客观性。更重要的是，当到了要做艰难决策的时候角色模型能发挥更大的作用，因为你在砍掉某个具体特性时，感觉受到不公

平待遇的不是那个发明它的人，而是某个角色模型。比如在莎拉这个例子中你可能会这样说：“我们说过莎拉比斯坦利更重要，所以这里我们需要为莎拉做优化。”

11.3.3　使用客观指标

在面试产品经理时我要求面试者讲述一个改变其他人想法的例子，他们几乎总是说起自己是如何通过事实来使其他人回心转意的。这样的产品经理应聘者很少会被录用，因为他们的故事必然会以失败收场。真是令人惋惜，但真实世界就是这样的，它可不怎么关心事实。你能从福克斯新闻“目前没有事实来支持这一推测”这样的报道方法中看到关于这个趋势的证据。如果同意事实没有观点重要，那你接下来就需要一套与观点打交道的方法。幸运的是我们有可用性测试和决策矩阵来帮忙。

可用性测试非常适合证明一个主观体验——用户界面是好是坏。你可以在第 3 章读到更多关于可用性测试的内容。

决策矩阵则是一个简单的图表，它可以帮助你做多选一的决策。表 11-1 列举了一个用于帮助我挑选宠物的决策矩阵。

表 11–1　宠物的决策矩阵

标准 \ 动物	猫	狗	兔
不掉毛	0	2	0
可搂抱	1	0	2
友好	1	2	0
总分	2	4	3

矩阵非常清晰地表明我应该买只狗。在建立了一系列评估标准并以此为基础集中评估了各个选项后，你将制作出一张清晰展现你的目标和优先级的图表。除此之外，由于团队可以从各个维度评估，讨论可以聚焦在更细微的元素上，因而没有人会觉得自己的意见全部打水漂了。换句话说，如果我认为“狗在可搂抱这块明显只有 1 分”，而你认为“像狗这样喜欢流口水的动物怎么可以去抱，0 分”，我们还是很有可能达成共识的，至少比争论猫和狗哪个更好更有可能达成共识，因为对我们来说最坏的情况

不过是多给狗 1 分或少给狗 1 分。这和争论猫和狗哪个好有本质区别。

如果时间充足，进一步使用决策矩阵是很有价值的。你可以增加一列“权重”或“优先级”，如表 11-2 所示，团队将不得不就各个决策标准的相对重要性进行讨论并取得共识。这非常有利于你们的团队成员归到一个频道上。以你自己为例，你可能会要比较剩余开发时间、可扩容性、可测试性以及其他所有好的、重要的标准，但你不可能既要可扩容性和可测试性又不投入更多的开发时间！而在我的例子中，我真不喜欢会掉毛的动物。

表 11–2 带权重的决策矩阵

动物 标准	权重	猫	狗	兔
不掉毛	2	0	2	1
可搂抱	1	1	0	2
友好	1.5	1	2	0
总分		2.5	7	4

嗯，史努比会是最好的选择。

第 12 章 胜在从容

12

SHIPPING GREATNESS

我曾想尽办法给交付裹上一层糖衣，但鄙人做不到啊！于是我只能把炮弹抛出来了：交付卓越的软件是件无比困难、压力巨大的事情。当然奖励也会十分丰厚。你付出的精力值得这样的回报。但附在这些待交付物上的压力真的大得很夸张。有时你会被要求独自去平衡几个互相竞争的优先级，有时你会被要求在半小时内完成需 3 个人做 1 小时的工作量。而且在整个项目过程中，你几乎肯定得面对新的特性需求、公司优先级的变化、职场政治以及普遍的不公正。但是等等，不止这些！你还有希望！这里有一些诀窍能帮助你应对这些交付生涯中的挑战。

长期以来，我和同事一道致力于交付事业，下面这些就是我收集并践行的经过实践检验的建议的精髓部分。我把它分为 5 个类别，希望能对你有所帮助。

- 如何平衡交付、质量和影响、团队这三者的关系，你才能交付一款卓越的软件。
- 如何应对随机情况，你才能继续按原定节奏交付一款卓越的软件。随机情况是指当你的管理层掷给你一个弧线球或者你的团队脱离正轨时出现的情况。随机情况是每个在谷歌或亚马逊工作的人都理解的词之一，因为它是与帮助团队专注于交付相对立的。
- 如何妥善地管好你的精力，你才能 1 个顶 3 个。
- 如何找人以及在什么时候找人，你才能让合适的人做合适的事。
- 如何咽下狗屎三明治，因为有时候你的确得咽下去。

12.1　如何平衡交付、质量和影响、团队这三者的关系

暂时假设你是一位 CEO（如果你确实是 CEO，请给你公司的全体员工购买这本书）。如果你的团队主管高兴地向外界宣布你们的新产品已准备好对外发布，但实际上它只是一坨热腾腾的屎，你不会认为这个团队或者这个团队主管是成功的。我也不会这么认为。所以这个主管将被解雇。

如果你手下的主管又快又好地完成了交付，但他依靠的是个人魅力或者一些迷惑人心的招数，因而团队成员在交付完成后变得精疲力竭、心灰意冷，你也会认为他是

失败的并解雇他。如果你不解雇他，董事会就会解雇你。

如果你手下的主管因为一时过分的成本削减和工期压力而砍掉大量特性，虽然没有 Bug 了，团队也能提前交付，但用户不会关心这样的产品，这个主管也逃脱不了被解雇的命运。

因此一个成功的团队主管必须会平衡团队、质量和影响、想要马不停蹄地交付的欲望这三者的关系。在你设法完成交付过程的各个阶段以及本书所提到的各个具体事项时，切记这个平衡（图 12-1 所示）。维持住它，你就能保住饭碗。你要知道你不可能面面俱到，但你可以保持平衡，它会帮助你掌控你的交付生涯。现在你还可以使用一些工具来检查你是否失去平衡，如团队周会的气氛、高中蒙羞测试以及你的产品需求文档。你仍然在解决一个大众所共有的大问题吗？

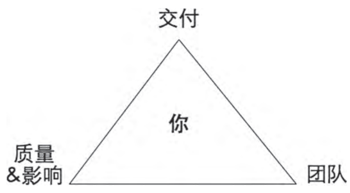


图 12-1　平衡的三角关系

12.2　如何应对随机情况

由于你不是老板，享受不了老板的特权，所以你的产品很可能被源源不断的出于好心却带来破坏的建议所伤害。不幸的是，当产品变得更重要更接近发布点时，你收到的建议也会变多，因为通过内部测试过程你的产品获得了更多的曝光。你无法摆脱这些建议，但你可以使它们不会干扰或“打乱”你的团队。随机情况感觉就像你在阶段 2 时有个人跑过来扔了一个 18 面的色子，然后说“情况变了，改去阶段 13 吧”。

这通常不是你的错。随机情况的来源有很多，最常见的就是新的特性需求，也可能是公司的优先级发生了变化，甚至还可能是你被强制接受某些大规模的基础设施变更。你必须要从容应对这些随机情况并设法照常推进业务。

为了应对新的特性需求，你可以创建一个简单的共享文档来集中记录它们。将文档对每个人开放，这种透明度会减少人们出于恐惧的关心和你收到的问题数量。当有人建议一个新特性时，对他说："太感谢你了！我把你的想法加到了我们的需求收集清单里，这是链接地址。"这个方法等同于开会时你把团队成员发表的想法记在白板上，记下反馈可以直截了当地表明你收到了（但不代表你答应了）。

当随机的建议来自管理层或投资人时，你可以使用版本 1 测试来帮忙打消这个想法。问："你想因为它而推迟发布版本 1 吗？"有时你会听到一个斩钉截铁的"是"。这时候你会全身上下都觉得惊慌失措。先别焦虑，还有希望。

当创始人或董事会说"是的，必须放在版本 1"时，你下一步就要把你的开发主管拉到一边告诉他发生了什么事情。询问开发主管这个变动会给工程造成什么影响。你需要从开发时间、系统设计、容量以及其他你所依赖的职能部门（如法务）等方面来评估成本。如果评估出来的成本很低，那就无须担心，直接把它加入到开发计划中吧。除非你已经非常接近发布时间了，这时候"说不"的协议自动生效（见第 7 章）。如果你的开发主管评估出来的成本很高，那就带着评估结果回到高层那儿，问："你愿意为了这个特性推迟 6 个礼拜发布并再购买 100 台服务器吗，或者我们可以在发布完成后再立即开发它？"这时你们可以就成本和收益进行真实、理性的交流。在成本评估前你没有任何理由退回任何一个需求，除非这个需求是不正当的。

如果这个建议真得非常奇葩但又必须放在版本 1，有时候你最该做的事情不是去反对。如果它是一个糟透了的主意，你的可信测试者们会告诉你的，而你的高层在畅所欲言的客户和实实在在的用户数据面前是很难再一意孤行的。所以遇到这些情况你可以和你的团队一起投入最小的精力并以最快的速度将这个特性部署到你的可信测试者那儿去，然后等待并观察。你也许会错，但如果你没错，就可以把这些数据带回给管理层并潇洒离去。

在评审会议上认真倾听能帮助你避免某些随机情况。切记，如果大老板说你应该做某件事，你几乎一定需要做这件事。如果你没有正面回应他的要求或者没有按照他所期望的进行交付，就会被打乱阵脚。不要存在侥幸心理认为老板们会忘了他们提出过这个需求。如果你在会上一字不漏地做了笔记，就能明确地知道老板的期望是什么，也就能够设法完成它们。这样做能大幅减少你被打乱阵脚的可能性。

当然，还有很多随机情况的来源是你无法控制的，比如公司优先级发生了变化。

深入探讨这些应对职场政治的诀窍和技巧超过了本书的范围。在我看来有些人天生就适合搞政治，而有的人脑袋里就没有一根政治的筋。我属于后一种人，所以我只能提供五条基本的策略来应对公司优先级的变化，这五条策略是按先后顺序排列的。切记，你的目标是交付，所以你需要工程团队当做什么事情都没有发生一样继续推进你们的产品。

1. *弄清楚公司优先级的变化是不是真的*。有时领导者声称公司优先级正在逐步改变，但公司却并不打算坚持到底或者缺少能力完成改变。那些知道变化是否为真的人们通常只和变化的发起者相差一个层级。找这些相差一个层级的人聊天并私下从他们那获得一些诚实的评价，不要记录。你可能会听到一些有别于官方说法的东西，并发现原来高层只是在设法安慰一些外部组织。你甚至还可能听到这个决策还没有最终敲定。如果是这种情况，你就可以照常向前推进，你的应对工作也可以暂时告一段落了。

2. *更改产品的表达方式以迎合变化*。有时候你不需要真的调整工程师的工作，而只是简单地更改产品的表达方式就能迎合新的优先级。更改你简短的演示文稿的表达方式并检查它是否承认了你懂得新的世界秩序会为你和你的团队赢得喘息的空间。这样成功迎合上了公司新的优先级，你就能照常向前推进。

3. *尽可能小地改动*。如果想照常推进发布且又必须进行改动，那么你需要尽量控制改动的规模。你改动越大，遇到的随机情况就会越多，发布的风险也就越大。请和你的工程团队合力找出一条成本低见效快的方法。通常快速地完成细微的事证明了你的团队的注意力是集中的，它还为你赢得了足够的时间去发布。事实上，如果能率先展示出适当的符合新的优先级的进展，你也许能为你的团队赢得超乎寻常的余地去按照你们自有的节奏向前推进，因为你清晰地表明了你能做好这份工作。请进行小的改动，然后回归正常的节奏。

4. *请求通融*。有时候请求通融是向前推进的最佳办法。如果你必须先进行需暂时中止交付的重大改动才能继续向前推进的话，那么得到全部人的通融就特别重要了。另一种请求通融的方式是请求在版本 2 中再响应公司新的优先级。你要用对付大老板提出新的想法的方式向人们解释新的优先级的响应成本。如果得到通融允许你把这些改动放到版本 2，你就应对成功了。又一次你可以照常向前推进。

5. *忍*。如果你试过所有这些方法都没有丝毫进展的话，我只能表示抱歉了。你将处于一个艰难的境地并不得不咽下狗屎三明治，很快我们就会讲什么是狗屎三明治。

请咽下狗屎三明治并撰写新的产品需求文档。切记重置你的时间表，并忘记这么做将对你的团队造成极不公平的后果。

所有这些策略也适用于应对大规模的基础设施变更。这里还有一个关于如何与你所依赖的基础设施打交道的经验法则：你需要坐在火车中部。基础设施项目就像一列驶进城里的大型的、轰轰作响的火车。启动这类项目需要很长时间，且多数情况下它们都声势浩大、一意孤行且极不讨人喜欢。如果你位于火车前列，即基础设施的早期使用者，你将遭遇 Bug、肥胖笨拙的婆娘和偶然出现的卡曼格 · 希亚跑车——没有一样是有趣的事情。所以你不会想成为一个重大基础设施项目的早期使用者。

另一方面，如果你挂在火车尾部则很有可能被敲打鞭策。管理层会问："你为什么还不移植到 X 呢？"你的系统很可能开始出现一些滑稽的故障。对你的旧的基础设施的支持将慢慢消失，总体上你将处在一个很不舒服的境地。所以你不会想挂在火车尾部。

你需要的是坐在火车中部——刚好靠近拥有鸡尾酒和椒盐卷饼的休闲车厢。找到两到三个可供参考他们进展的团队。当发现某个团队切换到新基础设施的过程相当轻松，那么你也是时候切换了。新基础设施的文档资料这时很可能比较全面了，但你还是可以请求那个成功切换的团队来指导你完成整个切换过程。你从用户那边获得的指导几乎总是比你从基础设施开发者那边获得的指导要好。

12.3 在交付过程中如何管理精力

我认识的一个亚马逊的工程经理曾用仅仅 20 分钟准备一个给高管汇报的重要演示稿。这不像他的风格，他曾用过连续一周的时间去解决生产环境的问题。我转过椅子问他："这是真的吗？"他耸耸肩，说："我还有个会，等我回来回答你。"

半小时后，他开完会回到座位上，然后坐下来说："好吧，我分配在演示稿上的时间是合理的。"这句话对我的职业生涯产生了巨大影响，它教导我必须学会为每个任务分配合理的时间。在这个例子中，他非常确定高管已经有了自己的决定，所以为什么要浪费大量时间做一个高管只会扫一眼的演示呢？还有很多交付相关的事情需要你去做，其中大部分本书都有详细说明，但你不可能做完所有事情。

亚马逊前全球大发现（Worldwide Discovery）副总裁金 · 雷切米勒曾跟我说："当我雇佣首位项目集经理进亚马逊时，我让她坐在我旁边，然后每天下班时我会说今

天我又没有做完所有事情。我这样做是因为我想让她明白项目集经理的工作是做不完的——但你需要做完今天必须做完的工作。”

因为不可能做完团队需要你做的所有事情，所以你需要优先做最应该做的事情，且不要在意没有做完所有事情。彼得・德鲁克在《卓有成效的管理者》中也呼应了这个观点。他说你首先应该做那些只有你能做的事情。这个工作方法有助于将你起到的作用最大化并防止你成为阻碍者。

举一个这个方法在谷歌发挥作用的小例子。在谷歌有个传统是费用报表的审批都非常快。优秀的管理者知道他们是那个任务的最关键的阻碍者，所以他们会立即处理它。

我过去总是很担心自己精力不够。所以我问埃里克・施密特我们是否有什么计划来帮助那些没日没夜加班的产品经理。埃里克说：“我认为很多产品经理首先在时间管理上就是低效的。”我感觉受到了侮辱，我认为我的工作做得够好了，而且我的确感觉到精力被牵扯得太分散了！但当我事后看到埃里克的工作量，再联想到我学过的首先做那些只有你能做的事情，合理分配每个既定任务的时间以及不要在意事情没有全部做完时，我认为他是对的。

如果你觉得学会高效利用时间和精力太难了，或者认识到这是一个你完全缺失的技能时，考虑从托尼・施瓦茨的能量工程中学会该如何做吧[①]。施瓦茨曾在《哈佛商业评论》上发表了一篇名为“管理你的精力，而不是时间”（Manage your energy, not your time）的卓越文章并随后围绕优化个人精力这个概念出版了一本书并开发了一套业务[②]。他主要提倡你应了解你的精力都花在哪儿了、如何优化你的精力分配以及如何储备更多的精力。很多卓有成效的管理者都开始采纳他的方法，因为它的确有用。有些 CEO 还会在办公室备有枕头，这确实有点疯狂，但看起来很有效率。

最后一个管理精力的技巧是制定工作日程表。在交付的过程中你的全部或者大部分时间都消耗在了会议上。这意味着你只有一丁点的时间来撰写产品需求文档、检查用户体验以及和你的工程团队讨论系统设计。而且由于一些或好或坏的原因，人们宁愿预定你的时间直到深夜，也不太会预定你的“工作时间”。因此托尼・施瓦茨建议你安排早上开始的一个半小时来攻坚你最困难的任务。一个半小时恰好可以让你不需

①http://www.theenergyproject.com/

②http://hbr.org/2007/10/manage-your-energy-not-your-time/ar/1

要休息并保持专注，而且早上是我们精力最充沛的时候。通常来说，你最困难的任务也是你最不想做并最可能拖延的任务，因此如果你是一个拖延症患者，制定工作日程表会是一个很不错的推动你去做事的方法。值得去尝试，不是吗？

12.4 如何把向上求援当成工具而非托词

我们已经证实你不是老板，但总有人是。而且这些爱发号施令的家伙可以成为很有价值的人！知道何时向上求援是一种重要的能力，就像你在幼儿园里学到的“向上”是“向上、向下、穿过”的重要组成部分一样。

当遇到下面任何一种情况时，你很可能需要向上求援。

- **你正设法为一个在某些方面比较愚蠢的管理层下达的命令辩护**。你不需要为那些愚蠢的想法辩护。或许终究还得你上，但最好先让老板们为他们那些愚蠢的想法辩护吧。
- **你真心不懂为什么你要做这些事情**。像这种情况，你最好能单独从大老板那里得到一个解释，因为大老板可能会更清楚一些。如果你通过邮件要求得到一个解释，请不要抄送给团队成员。
- **你不负责这个问题**。一个典型的需要你向上求援的情况是你对工程团队所做的决定存在技术方面的疑虑。大部分有工程背景的领导者都能凭直觉感受到火车正在滑出轨道。不要试着和你的技术主管正面对抗，让拥有合法授权并负有责任的工程经理来处理这个事情。除此之外，在这场管理者的危机中站在工程团队一边有助于你在团队中树立公平的形象。

  这里需要重点说明的是我并非主张你应事不关己高高挂起。如果有些事情你可以解决，你的团队也希望你去解决，那就去解决吧！在解决过程中请积极主动并坚定自信，不要被你的职位名称或职位描述所限制。但当你发现有个问题是其他人负责的，且那个人不喜欢你越俎代庖，那就向上求援吧。
- **你需要与之打交道的高管不愿接见你**。他们不愿接见你的原因可能是因为你的级别不够高，或者更可能是因为他们不知道你。但你的管理层或投资人很可能和这些讨厌的高管有现成的关系。他们经常一起开会（读作：打高尔夫）所以相互之间建立了信任。利用他们这种通过密切交往而建立起来的信任和理解作

为敲门砖是种高效的方法。

12.5　如何咽下狗屎三明治并生存下去

在交付产品的过程中你不可避免需要咽下一定数量的狗屎三明治。其中一些三明治来自于合作伙伴，一些来自于你的老板或投资人，还有一些来自于爱抱怨的工程师或竞争对手。你现在就需要接受这个事实：你团队主管的生涯中一定会经历一些令人厌恶的时刻。

我曾在亚马逊做某个项目时遇到一次危机，连我的高管都吓坏了。整个过程简直就是煎熬。客户纷纷给杰夫写邮件，而你知道他们肯定写的不是谢谢你之类，因为他们从来不写这个，他们只会在邮件中痛骂你。

帮助我在这场近乎职业生涯的大灾变中生存下来的是我从某个家伙那听到的一句话。当时我正在等待副总裁从电梯下来给我送上特别的狗屎三明治，一个坐我边上的同事说："这种事就是痛一会儿，然后就过去了。"这句话我始终铭记在心，并经常说给其他人听，可能说得比我记忆中的还要多。

你可以把这些三明治当作高中生活，生存下来才是重点。当收到一个三明治时你应保持微笑，咽下三明治，然后继续前行。谷歌产品总监，且属于最具大将之风的那群人之一的马特·格罗茨巴赫曾对我说："别人越慌乱你就应该越镇定。"所以咽下三明治，保持微笑，并且记住它是什么滋味，这样你就会少准备些这种三明治给你的团队。

还有一个小小的关于交付的告诫。在某些恶劣的环境中，你可能发现你无时无刻不在吃这些三明治。这不是一个好的信号。电脑朋客类科幻作品的典范《神经漫游者》的作者、技术文化的卓越理解者威廉·吉布森曾说："在你断定自己精神抑郁或缺乏自信之前，先确定你实际上没有被一群白痴包围。"有时候你被要求咽下的三明治的数量和质量都是直接来自和你一起工作的白痴们。当碰到这种情况时，你就该换个地方，换个产品去交付了。

第 13 章 再度启航 13

SHIPPING GREATNESS

在经历了一个主要的产品开发周期的戏剧性、痛苦和艰辛后，你很容易只去关注软件的不足。但既然飞行员可以说“能安全走出机舱的着陆就是最好的着陆”，你也可以说能交付的软件就是最好的软件。交付才是关键。那么在完成交付后会发生什么呢？

REI.com 的项目集经理亚伦·艾布拉姆斯说过：“对项目集经理来说有两个美好的时刻：拿到项目的时刻和交付的时刻。”希望你在项目彻底交付后再举行庆祝活动（见第 7 章）。而现在你该准备迎接另一个美好时刻——拿到新项目的时刻。

在开始新项目之前，先回过头来审视一下周边的环境。软件从来没有做完一说。你需要问问自己是应该开始做软件的第二版，还是开始做些新的东西。某些情况下你会发现对产品的投入所带来的回报正在递减。我所说的投入是指你的时间。时间构成的风险资本是最具利用价值的资产之一。

如果你各方面都表现不凡——按照本书的指导来做事你就能变成这样，就永远可以选择下一个项目。考虑每种选择对你的业务或个人意味着什么。或许是时候换一个团队或换一个产品了，或许你因为理解错了客户的需求而需要重构刚刚交付的产品，或许你只是简单地喜欢和那个工程经理小伙伴一起工作。评估维度有很多，你需要花些时间来考虑。

你在定义上个产品时所有运用过的技能在这个反思阶段都是有意义的。你可以想想客户问题、你的业务的独特优势以及个人的强项。每个人都有最适合自己的选择，即便下个项目已经被内定，你也依然可以选择聚焦在何处以及如何实现它。

一个出色的主管不会什么东西都交付，他只交付正确的软件。如果你想让别人看到一个卓越的、专业的你，就必须慎重考虑时间的投入方式。

在决定接下来做什么后，就得过渡到下个项目中。亚伦·艾布拉姆斯也参加过铁人三项赛（铁人三项赛是我认为的少数几件能在难度上超过在试图交付过程中保住饭碗的事情之一）。他这样说：“过渡——你知道的，从游泳到骑自行车，或者从骑自行车到跑步，是困难的部分。”我倒觉得铁人三项赛中任何部分都是困难的，但他提出了一个很好的观点。在软件行业中，过渡到一个新的项目通常是很有挑战性的。

项目过渡之所以具有挑战性，是因为它就像铁人三项赛一样，你必须中断之前的

做事方式（如专注于细节、预发布）然后开始新的做事方式（如参与头脑风暴和策略游戏）。它的另一个挑战性还在于你需要设法做好两份工作。第一份工作是维护已经在生产环境中的软件，它几乎肯定会遇到一些早期发展过程中的困难。第二份工作是启动新项目。大部分新项目在启动时都需要巨大的精力去推动，并在推动不利时需要以坚韧的精神去承受不可避免的迎头痛击。

处在过渡时期是非常艰苦的，所以请缩短过渡时间，把它缩短到你都觉得不可思议的程度。如果曾经注意到当你休长假时办公室里的各项工作进展都比你预期得要好，你就会发现在你离开之前的项目情况也会是这样的。团队很可能只减速一会儿。他们未来几乎肯定会做一些让你拍打额头的事情，或者做一些让你在倒咖啡时喋喋不休地抱怨的决定，但这些再也不是该你头疼的问题了。更恶劣的是你还可能看到你的产品在它的新主管带领下出现在超级碗的广告中。我就遇到了这样的事，不过哥们，你觉得我会后悔当初离开的决定吗！

最终所有这些戏剧性都会消失，因为你正忙于交付一些新的东西。你虽然交付了他们软件的第一个版本，但未来已经与你无关。祝福你的前团队然后回到工作上来。检查你的使命和策略，然后开始撰写下一篇新闻稿。

祝你好运！

克里斯·范德·梅

2012 年于华盛顿州西雅图

附录 1

十大交付原则

1. 你不是来当老板的——团队主管是仆人，他们存在的目的就是为了伺候工程团队。
2. 从用户角度出发。
3. 用独特的方法解决很多人都有的大问题。
4. 坏的消息就是好的消息。——杰克 · 韦尔奇
5. 先寻求理解，再寻求被理解。——史蒂芬 · 柯维
6. 构建最简洁的可用的产品。
7. 交付手中有的，而非脑中想的。
8. 无法测量的东西也就无法提升。——开尔文勋爵
9. 你不可能做完所有工作，所以你应首先做那些只有你能做的工作。
10. 永远走在交付的康庄大道上。

附录 2

团队不可或缺的工件

在管理产品开发的过程中，你会创建很多文档、指南、核查清单以及其他工件。下面这张列表是贯穿产品生命周期中你可能期望创建的工件的略述。你很可能需要创建以下所有工件，所以它们没有按照特别的顺序排列。你能从 www.shippinggreatness.com 上下载到其中一些工件的模板。

- 轮值表——将寻呼器号码和手机号码的清单复制到一张钱包大小的纸上。
- 使用 Wiki 搭建的“联络簿”，用于遇到故障、突发事件或问题时寻找相关负责人。这个列表应该包括法务、公关、市场、产品团队、工程团队和网络运维（或者任何负责生产基础设施的部门）的负责人和联系信息。
- 描述使命的语句。
- 关于未来两年的清晰策略。
- 一页简要说明产品的人物 / 事件 / 原因 / 时间 / 方法的文档。
- 产品需求文档，或者叫功能规格说明。
- 新闻稿。
- 线框原型图或者餐巾纸草图。
- 内部 FAQ 文档，其中部分问答打上外部 FAQ 标签以作为客户支持内容的原始素材。
- 沟通文档，包括关键信息、有潜在危险的问题和对这些问题的回应。
- 发布时穿的 T 恤衫。
- 包含测试时间的开发计划表。

- 未来两年的路线图。
- 内部用户列表和迁移时间表（适用于基础设施项目）。
- 可信测试者列表（适用于面向外部的产品）。
- 特性需求列表，并将内部和外部客户中呼声最高的三个特性需求高亮。
- 开放问题列表，并清晰标记这些问题的状态。
- 进行中的会议纪要。最好建一个文档保存项目所有的历史会议纪要。
- 发布计划或发布规程。
- 记录什么特性在什么时间发布的生成变更列表。在排查客户问题时特别有用。
- 对增长预测和硬件配置需求提前进行计划的生产设计文档。
- 专利注册文件、商标注册文件和版权申明文件。
- 隐私说明。
- 出色的数据指标——包含内部的状态面板和一些供外部消费的清洗过的数据指标。
- 为幻灯片、演示、评审、发布准备的产品截图。
- 团队本季度目标以及上季度目标完成情况。
- Bug 状态面板和阻碍每个发布的 Bug 列表。
- 错误原因报告或事后调查报告。
- 会议纪要和团队周会、用户界面评审、产品评审、工程评审、Bug 处理、法务评审、业务拓展周会以及客户支持碰头会的时间计划表。

附录 3

参考资料及延伸阅读

产品定义

Guy Kawasaki：*The Art of the Start: The Time-Tested, Battle-Hardened Guide for Anyone Starting Anything*。

Eric Ries：《精益创业：新创企业的成长思维》。

驾驭管理

Larry, Bossidy, Ram Charan：《执行：如何完成任务的学问》。

Peter F. Drucker：《卓有成效的管理者》。

Robert Fisher, William L. Ury, Bruce Patton：《谈判力》。

John P. Kotter：*Leading Change*。

工程管理

DeMarco, Tom, Timothy Lister：《人件（第 2 版）》。

用户体验

Pruitt, John, Tamara Adlin：*The Persona Lifecycle: Keeping People in Mind Throughout Product Design*。

Edward R Tufte：*The Visual Display of Quantitative Information, Second Edition*。

Robin Williams：《写给大家看的设计书》。

指标

Eliyahu M. Goldratt, and Jeff Cox：《目标：简单而有效的常识管理》。

沟通

Edward De Bono：《六顶思考帽》。

Schwartz, Tony, Catherine McCarthy："Manage Your Energy, Not Your Time"，《哈佛商业评论》。

http://hbr.org/2007/10/manage-your-energy-not-your-time/ar/1。

版 权 声 明

“在移动互联网时代，产品的用户体验比概念更重要。我们要靠技术给用户带来体验上的优势，而不是靠概念。本书不仅着墨于产品的极致体验，也为互联网从业者总结了一整套关于产品迭代、发布、市场推广等的简单框架，适合实际操作，且易于学习。”

——傅盛，猎豹移动CEO

“这是一本不错的产品经理或项目经理的入门级手册，对于一些刚选择做产品经理的人来说，能成为指路明灯。书中用到了不少作者在谷歌和亚马逊工作时的例子，能帮忙读者更好地理解产品交付的全过程。”

——陶伟华，360手机助手高级总监

“本书堪称软件项目管理人士的最佳基本功指南，建议所有软件开发人员都看看。”

——Matt Shobe，全球最大的RSS托管服务网站FeedBurner联合创始人

“Mey在这本书里汇总了世界一流产品开发流程和其中的经验总结，实践性超强，值得所有软件开发主管借鉴。”

——Mike Smith，迪斯尼互动CTO

“对于打算做一个项目并且只有时间读一本书来了解该怎么做的职场人士来说，这本书就是不二之选。”

——Lars Thorup，资深软件咨询顾问，ZeaLake公司CEO

“如果你有产品的想法，知道该如何做，并且迫不及待要开工，那么一定要看看这本书。”

——Tamara Adlin，*The Persona Lifecycle*作者

图灵社区：iTuring.cn
热线：(010)51095186转600

O'REILLY®
oreilly.com.cn

分类建议 计算机/互联网/产品经理
经济管理/团队管理

人民邮电出版社网址：www.ptpress.com.cn

O'Reilly Media, Inc.授权人民邮电出版社出版

此简体中文版仅限于中国大陆（不包含中国香港、澳门特别行政区和中国台湾地区）销售发行
This Authorized Edition for sale only in the territory of People's Republic of China (excluding Hong Kong, Macao and Taiwan)

ISBN 978-7-115-34931-6

ISBN 978-7-115-34931-6
定价：49.00元

欢迎加入

图灵社区

电子书发售平台

电子出版的时代已经来临，在许多出版界同行还在犹豫彷徨的时候，图灵社区已经采取实际行动拥抱这个出版业巨变。相比纸质书，电子书具有许多明显的优势。它不仅发布快，更新容易，而且尽可能采用了彩色图片（即使有的书纸质版是黑白印刷的）。读者还可以方便地进行搜索、剪贴、复制和打印。

图灵社区进一步把传统出版流程与电子出版业务紧密结合，目前已实现作译者网上交稿、编辑网上审稿、按章发布的电子出版模式。这种新的出版模式，我们称之为“敏捷出版”，它可以让读者以较快的速度了解到国外最新技术图书的内容，弥补以往翻译版技术书“出版即过时”的缺憾。同时，敏捷出版使得作、译、编、读的交流更为方便，可以提前消灭书稿中的错误，最大程度地保证图书出版的质量。

开放出版平台

图灵社区向读者开放在线写作功能，协助你实现自出版的梦想。你可以联合二三好友共同创作一部技术参考书，以免费或收费的形式提供给读者，这极大地降低了出版的门槛。成熟的书稿，有机会入选出版计划，同时出版纸质书。

图灵社区引进出版的外文图书，都将在立项后马上在社区公布。如果有意翻译哪本图书，欢迎来社区申请。只要通过试译的考验，即可签约成为图灵的译者。当然，要想成功地完成一本书的翻译工作，是需要有坚强的毅力的。

读者交流平台

在图灵社区，读者可以十分方便地写作文章、提交勘误、发表评论，以各种方式与作译者、编辑人员和其他读者进行交流互动。提交勘误还能够获赠社区银子。欢迎大家积极参与社区开展的访谈、审读、评选等多种活动，赢取银子，可以换书哦！